

[illegible]

```

LL          IIIIII          SSSSSSSS
LL          IIIIII          SSSSSSSS
LL          II             SS
LL          II             SS
LL          II             SS
LL          II             SS
LL          II             SSSSSS
LL          II             SSSSSS
LL          II             SS
LL          II             SS
LL          II             SS
LL          II             SS
LLLLLLLLLLLL IIIIII          SSSSSSSS
LLLLLLLLLLLL IIIIII          SSSSSSSS

```

EXE
V04

[illegible]

EXECUTE_REQUEST: Procedure

Returns(Fixed Binary(31))
Options(Ident('V04-000'));

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48
```

```
/*  
*****  
/* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY  
/* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.  
/* ALL RIGHTS RESERVED.  
/*  
/* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  
/* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE  
/* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER  
/* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY  
/* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY  
/* TRANSFERRED.  
/*  
/* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE  
/* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT  
/* CORPORATION.  
/*  
/* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS  
/* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.  
/*  
*****  
*/  
/*  
/*++  
/* FACILITY: MONITOR Utility  
/*  
/* ABSTRACT: EXECUTE_REQUEST Routine.  
/*  
/*          Called from MONMAIN routine to execute a single  
/*          MONITOR request.  
/*  
/* ENVIRONMENT:  
/*  
/*          Unprivileged user mode,  
/*          except for certain collection routines which  
/*          run in EXEC or KERNEL mode to access system  
/*          data bases.  
/*  
/* AUTHOR: Thomas L. Cafarella, April, 1981  
/*
```

```
49 | 1 | /*
50 | 1 | /* MODIFIED BY:
51 | 1 | /*
52 | 1 | /* V03-026 TLC1091 Thomas L. Cafarella 08-Aug-1984 15:00
53 | 1 | /* Save summary buffer data for only those classes requested;
54 | 1 | /* exclude extra classes collected in support of SYSTEM class.
55 | 1 | /*
56 | 1 | /* V03-025 TLC1090 Thomas L. Cafarella 02-Aug-1984 15:00
57 | 1 | /* Correct ACCVIOs in SYSTEM and PROCESSES classes.
58 | 1 | /*
59 | 1 | /* V03-024 TLC1086 Thomas L. Cafarella 24-Jul-1984 14:00
60 | 1 | /* Make top summary work for SYSTEM class.
61 | 1 | /*
62 | 1 | /* V03-023 TLC1085 Thomas L. Cafarella 22-Jul-1984 14:00
63 | 1 | /* Calculate scale values for Free and Modified List bar graphs.
64 | 1 | /*
65 | 1 | /* V03-022 TLC1082 Thomas L. Cafarella 23-Jul-1984 11:00
66 | 1 | /* Always save data in summary buffers, even when only
67 | 1 | /* one collection.
68 | 1 | /*
69 | 1 | /* V03-021 TLC1072 Thomas L. Cafarella 17-Apr-1984 11:00
70 | 1 | /* Add volume name to DISK display.
71 | 1 | /*
72 | 1 | /* V03-020 TLC1068 Thomas L. Cafarella 13-Apr-1984 14:00
73 | 1 | /* Fix bug causing a garbage heading display.
74 | 1 | /*
75 | 1 | /* V03-019 PRS1019 Paul R. Senn 11-Apr-1984 16:00
76 | 1 | /* Fix SYSTEM class /SUMMARY and change SYSTEM default interval.
77 | 1 | /*
78 | 1 | /* V03-018 TLC1060 Thomas L. Cafarella 12-Mar-1984 11:00
79 | 1 | /* Make multi-file summary work for homogeneous classes.
80 | 1 | /*
81 | 1 | /* V03-018 TLC1059 Thomas L. Cafarella 20-Mar-1984 11:00
82 | 1 | /* When re-recording, include input revision level
83 | 1 | /* in output file.
84 | 1 | /*
85 | 1 | /* V03-018 TLC1057 Thomas L. Cafarella 22-Mar-1984 15:00
86 | 1 | /* Eliminate node name from heading for multi-file summary.
87 | 1 | /*
88 | 1 | /* V03-018 TLC1056 Thomas L. Cafarella 22-Mar-1984 11:00
89 | 1 | /* Disable journaling classes and exclude class which is disabled.
90 | 1 | /*
91 | 1 | /* V03-017 PRS1011 Paul R. Senn 29-Feb-1984 14:00
92 | 1 | /* add /FLUSH_INTERVAL qualifier
93 | 1 | /*
94 | 1 | /* V03-016 TLC1052 Thomas L. Cafarella 17-Feb-1984 11:00
95 | 1 | /* Add multi-file summary capability.
96 | 1 | /*
97 | 1 | /* V03-015 TLC1051 Thomas L. Cafarella 11-Jan-1984 11:00
98 | 1 | /* Add consecutive number to class header record.
99 | 1 | /*
100 | 1 | /* V03-015 PRS1003 Paul R. Senn 9-Jan-1984 10:00
101 | 1 | /* Fix 1 bit parameter passing problem on call to DISP_TEMPLATE.
102 | 1 | /*
103 | 1 | /* V03-015 PRS1002 Paul R. Senn 29-Dec-1983 16:00
104 | 1 | /* GLOBALDEF VALUE symbols must now be longwords;
```



```
105 | 1 | /* Use %REPLACE rather than GLOBALDEF VALUE for any equated
106 | 1 | /* symbols which are not 4 bytes in length;
107 | 1 | /*
108 | 1 | /*
109 | 1 | /* V03-015 PRS1001 Paul R. Senn 27-Dec-1983 16:00
110 | 1 | /* Make default interval = 6 for ALL classes Pseudo-class
111 | 1 | /* live requests.
112 | 1 | /*
113 | 1 | /* V03-015 PRS1000 Paul R. Senn 15-Dec-1983 16:00
114 | 1 | /* For cases where one display event may involve multiple
115 | 1 | /* screens of data (such as PROCESSES and homogeneous
116 | 1 | /* classes), make the wait between screens = VIEWING_TIME,
117 | 1 | /* instead of a constant of 2 seconds.
118 | 1 | /*
119 | 1 | /* V03-014 TLC1050 Thomas L. Cafarella 06-Dec-1983 11:00
120 | 1 | /* Change directory information in DLOCK class.
121 | 1 | /*
122 | 1 | /* V03-013 TLC1047 Thomas L. Cafarella 09-Sep-1983 10:00
123 | 1 | /* De-establish CTRL/W handler to get back AST quota.
124 | 1 | /*
125 | 1 | /* V03-012 SPC0007 Stephen P. Carney 24-Jun-1983 16:00
126 | 1 | /* Add execute command file handling in CTRLZ routine.
127 | 1 | /*
128 | 1 | /* V03-011 TLC1042 Thomas L. Cafarella 19-Jun-1983 15:00
129 | 1 | /* Add /ITEM qualifier for homogeneous classes.
130 | 1 | /*
131 | 1 | /* V03-011 TLC1039 Thomas L. Cafarella 15-Jun-1983 15:00
132 | 1 | /* Add DECnet node name to heading.
133 | 1 | /*
134 | 1 | /* V03-011 TLC1037 Thomas L. Cafarella 14-Jun-1983 19:00
135 | 1 | /* Perform FLUSH after writing record (instead of before).
136 | 1 | /*
137 | 1 | /* V03-011 SPC0005 Stephen P. Carney 10-Jun-1983 15:00
138 | 1 | /* Make the playback/record file read-shareable
139 | 1 | /*
140 | 1 | /* V03-010 TLC1035 Thomas L. Cafarella 06-Jun-1983 15:00
141 | 1 | /* Add homogeneous class type and DISK class.
142 | 1 | /*
143 | 1 | /* V03-010 TLC1033 Thomas L. Cafarella 30-May-1983 16:00
144 | 1 | /* Don't clear screen for CTRL/Z.
145 | 1 | /*
146 | 1 | /* V03-009 TLC1032 Thomas L. Cafarella 27-May-1983 15:00
147 | 1 | /* Modify file structure level ID for LOCK class change.
148 | 1 | /*
149 | 1 | /* V03-008 SPC0002 Stephen P. Carney 22-Apr-1983 14:00
150 | 1 | /* Modify file structure level ID for new ACPACHE class.
151 | 1 | /*
152 | 1 | /* V03-007 TLC1028 Thomas L. Cafarella 14-Apr-1983 16:00
153 | 1 | /* Add interactive user interface.
154 | 1 | /*
155 | 1 | /* V03-007 TLC1027 Thomas L. Cafarella 14-Apr-1983 16:00
156 | 1 | /* Enhance file compatibility features.
157 | 1 | /*
158 | 1 | /* V03-006 TLC1022 Thomas L. Cafarella 12-Jul-1982 16:00
159 | 1 | /* Change recording file structure level since new classes
160 | 1 | /* (JOURNALING and RECOVERY) are now defined.
161 | 1 | /*
```

EXECUTE_REQUEST
V04-000

H 6
16-SEP-1984 02:15:13
5-SEP-1984 15:10:53

VAX-11 PL/I X2.1-273
DISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PLI;1 (2) Page 4

162	1	/*	V03-005	TLC1016	Thomas L. Cafarella	02-Apr-1982	16:00
163	1	/*		Replace references to EXESGQ_SYSTIME with \$GETTIM calls.			
164	1	/*					
165	1	/*	V03-005	TLC1014	Thomas L. Cafarella	01-Apr-1982	13:00
166	1	/*		Correct attached processor time reporting for MODES class.			
167	1	/*					
168	1	/*	V03-005	TLC1013	Thomas L. Cafarella	31-Mar-1982	09:00
169	1	/*		Do not clear TOP box until it fills with data.			
170	1	/*					
171	1	/*	V03-005	TLC1012	Thomas L. Cafarella	30-Mar-1982	13:00
172	1	/*		Display user's comment string on screen line 5.			
173	1	/*					
174	1	/*	V03-005	TLC1011	Thomas L. Cafarella	29-Mar-1982	20:00
175	1	/*		Move system service names for SSERROR msg to static storage.			
176	1	/*					
177	1	/*	V03-004	TLC1009	Thomas L. Cafarella	29-Mar-1982	01:00
178	1	/*		Get current time when other times are converted.			
179	1	/*					
180	1	/*	V03-004	TLC1008	Thomas L. Cafarella	28-Mar-1982	21:00
181	1	/*		Fix to display first and last PROCESSES records on playback.			
182	1	/*					
183	1	/*	V03-004	TLC1006	Thomas L. Cafarella	28-Mar-1982	13:00
184	1	/*		Add checks to skip data display on CTRL-C during template.			
185	1	/*					
186	1	/*	V03-003	TLC1003	Thomas L. Cafarella	23-Mar-1982	13:00
187	1	/*		Fix up module headers.			
188	1	/*					
189	1	/*	V03-002	TLC1002	Thomas L. Cafarella	20-Mar-1982	13:00
190	1	/*		Change PROCESSES display from scroll-style to page-style to			
191	1	/*		make it terminal-independent.			
192	1	/*					
193	1	/*		Move collection event flag to REQUEST.PLI for consolidation.			
194	1	/*					
195	1	/*	V03-001	TLC1001	Thomas L. Cafarella	16-Mar-1982	13:00
196	1	/*		Add CTRL-W screen refresh support.			
197	1	/*					
198	1	/*--					
199	1	/*					
200	1						


```
201 | 1 | /*
202 | 1 | /*
203 | 1 | /*
204 | 1 | /*
205 | 1 | /*
206 | 1 | /*
207 | 1 | /*/
208 | 1 |
209 | 1 | %INCLUDE      MONDEF;          /* Monitor utility structure definitions */
977 | 1 | %INCLUDE      $CHFDEF;         /* Condition handler facility definitions */
997 | 1 | %INCLUDE      $STSDEF;         /* Status value definitions */
1014 | 1 |
1015 | 1 | /*
1016 | 1 | /*
1017 | 1 | /*
1018 | 1 | /*
1019 | 1 | /*
1020 | 1 | /*
1021 | 1 | /*/
1022 | 1 |
1023 | 1 | %INCLUDE      SYSS$DCLAST;      /* $DCLAST system service */
1031 | 1 | %INCLUDE      SYSS$SETAST;      /* $SETAST system service */
1037 | 1 | %INCLUDE      SYSS$CLREF;       /* $CLREF system service */
1043 | 1 | %INCLUDE      SYSS$SETEF;       /* $SETEF system service */
1049 | 1 | %INCLUDE      SYSS$READEF;      /* $READEF system service */
1056 | 1 | %INCLUDE      SYSS$SETIMR;      /* $SETIMR system service */
1065 | 1 | %INCLUDE      SYSS$CANTIM;      /* $CANTIM system service */
1072 | 1 | %INCLUDE      SYSS$ASCTIM;      /* $ASCTIM system service */
1081 | 1 | %INCLUDE      SYSS$WAITFR;      /* $WAITFR system service */
1087 | 1 | %INCLUDE      SYSS$WFLOR;       /* $WFLOR system service */
1094 | 1 | %INCLUDE      SYSS$PUTMSG;      /* $PUTMSG system service */
1102 | 1 |
```

EXTERNAL STORAGE DEFINITIONS

```
1103 1 /*
1104 1 /*
1105 1 /*
1106 1 /*
1107 1 /*
1108 1 /*
1109 1 /*/
1110 1
1111 1 Declare
1112 1 ST_LEVEL_CUR CHAR(8) GLOBALREF; /* Current MONITOR recording file structure level */
1113 1
1114 1 Declare
1115 1 MAX_CLASS_NO FIXED BINARY(31) GLOBALREF VALUE, /* Maximum defined class number */
1116 1 CLASSTABLE FIXED BINARY(31) GLOBALREF VALUE, /* Addr of table of class names & numbers */
1117 1 VTWIDTH FIXED BINARY(31) GLOBALREF VALUE, /* Width of video terminal */
1118 1 VTHEIGHT FIXED BINARY(31) GLOBALREF VALUE, /* Height of video terminal */
1119 1 MAX_REC_SIZE FIXED BINARY(31) GLOBALREF VALUE, /* Max record size for PLAYBACK & RECORD files */
1120 1 PROCS_CLSNO FIXED BINARY(31) GLOBALREF VALUE, /* PROCESSES class number */
1121 1 STATES_CLSNO FIXED BINARY(31) GLOBALREF VALUE, /* STATES class number */
1122 1 MODES_CLSNO FIXED BINARY(31) GLOBALREF VALUE, /* MODES class number */
1123 1 SYSTEM_CLSNO FIXED BINARY(31) GLOBALREF VALUE; /* SYSTEM class number */
1124 1
1125 1 Declare
1126 1 CDBPTR POINTER GLOBALREF, /* Pointer to CDB (Class Descriptor Block) */
1127 1 C POINTER DEFINED(CDBPTR), /* Synonym for CDBPTR */
1128 1 MRBPTR POINTER GLOBALREF, /* Pointer to MRB (Monitor Request Block) */
1129 1 M POINTER DEFINED(MRBPTR), /* Synonym for MRBPTR */
1130 1 MCAPTR POINTER GLOBALREF, /* Pointer to MCA (Monitor Communication Area) */
1131 1 MC POINTER DEFINED(MCAPTR), /* Synonym for MCAPTR */
1132 1 SPTR POINTER GLOBALREF; /* Pointer to SYI (System Information Area) */
1133 1
1134 1 Declare
1135 1 MFSPTR POINTER GLOBALREF; /* Pointer to Multi-File Summary Block (MFS) */
1136 1
1137 1 Declare
1138 1 DISPLAYING BIT(1) ALIGNED GLOBALREF; /* YES=> display output is active */
1139 1
1140 1 Declare
1141 1 CTRL_CZ_CHAN FIXED BINARY(31) GLOBALREF, /* Channel number for CTRL-C and CTRL-Z */
1142 1 CTRL_W_CHAN FIXED BINARY(31) GLOBALREF; /* Channel number for CTRL-W */
1143 1
1144 1 Declare
1145 1 EXESGL_MP POINTER GLOBALREF, /* Pointer to multiprocessing data structures */
1146 1 SGN$GW_MAXPRCCT FIXED BINARY(15) GLOBALREF, /* MAXPROCESSCNT SYSGEN parameter value */
1147 1 PFN$GL_PHYPGCNT FIXED BINARY(31) GLOBALREF, /* Balance set memory size (in pages) */
1148 1 MPW$GW_HILIM FIXED BINARY(15) GLOBALREF; /* MPW_HILIMIT SYSGEN parameter value */
1149 1
1150 1 Declare
1151 1 SETIMR_STR FIXED BINARY(7) GLOBALREF, /* Count byte for $SETIMR cstring */
1152 1 DCLAST_STR FIXED BINARY(7) GLOBALREF, /* Count byte for $DCLAST cstring */
1153 1 SCHDWK_STR FIXED BINARY(7) GLOBALREF, /* Count byte for $SCHDWK cstring */
1154 1 READEP_STR FIXED BINARY(7) GLOBALREF, /* Count byte for $READEP cstring */
1155 1 CLREF_STR FIXED BINARY(7) GLOBALREF; /* Count byte for $CLREF cstring */
1156 1
1157 1
```



```
1158 | 1 | /*
1159 | 1 | /*
1160 | 1 | /*
1161 | 1 | /*
1162 | 1 | /*
1163 | 1 | /*
1164 | 1 | /*/
1165 | 1 |
1166 | 1 | Declare
1167 | 1 | MON_ERR ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE), /* MONITOR MACRO-32 routine to log synchronous error
1168 | 1 | SIGNAL_MON_ERR ENTRY, /* MONITOR MACRO-32 routine to signal MONITOR errors
1169 | 1 | READ_INPUT ENTRY (FIXED BINARY(31)), /* MONITOR routine to read an input (playback) file
1170 | 1 | ESTAB_CTRLZ ENTRY RETURNS(FIXED BINARY(31)), /* MONITOR MACRO-32 routine to set up CTRL-C and CTR
1171 | 1 | ESTAB_CTRLW ENTRY RETURNS(FIXED BINARY(31)), /* MONITOR MACRO-32 routine to set up CTRL-W handler
1172 | 1 | DISPLAY_INIT ENTRY RETURNS(FIXED BINARY(31)), /* MONITOR MACRO-32 routine to do display init */
1173 | 1 | SUMMARY_INIT ENTRY RETURNS(FIXED BINARY(31)), /* MONITOR MACRO-32 routine to do summary init */
1174 | 1 | COLLECTION_EVENT ENTRY, /* MONITOR routine to perform collection */
1175 | 1 | QUAD_LT_QUAD ENTRY (BIT(64) ALIGNED, BIT(64) ALIGNED) /* MONITOR MACRO-32 unsigned quadword compare routin
1176 | 1 | RETURNS(BIT(1)),
1177 | 1 | QUAD_EQ_0 ENTRY (BIT(64) ALIGNED) RETURNS(BIT(1)), /* MONITOR MACRO-32 quadword compare to 0 routine */
1178 | 1 | CVT_TO_DELTA ENTRY (FIXED BINARY(31), BIT(64) ALIGNED), /* MONITOR MACRO-32 rtn to convert to delta time */
1179 | 1 | MPCHECK ENTRY RETURNS(BIT(1)), /* MONITOR MACRO-32 rtn to check for MP capability */
1180 | 1 | COMPUTE_BOOTTIME ENTRY RETURNS(FIXED BINARY(31)), /* MONITOR MACRO-32 rtn to compute system time at bo
1181 | 1 | CLUS_NET_INFO ENTRY RETURNS(FIXED BINARY(31)); /* MONITOR MACRO-32 rtn to get cluster & net info */
1182 | 1 |
1183 | 1 | Declare
1184 | 1 | DISP_TEMPLATE ENTRY (POINTER, BIT(1) ALIGNED) /* Rtn to display the template */
1185 | 1 | RETURNS (FIXED BINARY(31)),
1186 | 1 | DISPLAY_PUT ENTRY(ANY, FIXED BINARY(31), ANY, ANY) /* MACRO-32 rtn to put a display string */
1187 | 1 | OPTIONS(VARIABLE)
1188 | 1 | RETURNS(FIXED BINARY(31)),
1189 | 1 | FILL_DISP_BUFF ENTRY (POINTER, BIT(64) ALIGNED) /* MACRO-32 Fill display buffer routine */
1190 | 1 | RETURNS (FIXED BINARY(31)),
1191 | 1 | DISPLAY_HOMOG ENTRY (POINTER) /* MACRO-32 rtn to display homog class data */
1192 | 1 | RETURNS (FIXED BINARY(31)),
1193 | 1 | DISPLAY_PROCS ENTRY (POINTER, BIT(64) ALIGNED) /* MACRO-32 rtn to display processes */
1194 | 1 | RETURNS (FIXED BINARY(31)),
1195 | 1 | DISPLAY_TOP ENTRY (POINTER) /* MACRO-32 rtn to display top processes */
1196 | 1 | RETURNS (FIXED BINARY(31));
1197 | 1 |
```

1198	1
1199	1
1200	1
1201	1
1202	1
1203	1
1204	1
1205	1
1206	1
1207	1
1208	1
1209	1
1210	1
1211	1
1212	1
1213	1
1214	1
1215	1
1216	1
1217	1
1218	1
1219	1
1220	1
1221	1
1222	1

/會
 /會
 /會
 /會
 /會
 /會
 /會

MESSAGE DEFINITIONS

Declare	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_SSERROR	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_DISPERR	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_BEGNLEND	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_HIB	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_CLASNP	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_CLASUNK	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_NOCLASS	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_CLASDISAB	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_BEGAN	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_PREMEOF	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_INVINPFIL	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_UNSTLEV	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_NOOUTPUT	FIXED	BINARY(31)	GLOBALREF	VALUE
MNRS_UNEXPERR	FIXED	BINARY(31)	GLOBALREF	VALUE


```
1223 1 /*
1224 1 /*
1225 1 /*
1226 1 /*
1227 1 /*
1228 1 /*
1229 1 /*
1230 1 /*
1231 1 Declare
1232 1 HEADER_TYPE FIXED BINARY(15) GLOBALDEF INIT(128), /* Type code for MONITOR recording file header */
1233 1 SYI_TYPE FIXED BINARY(15) GLOBALDEF INIT(129), /* Type code for MONITOR recording file sys info rec
1234 1 DISP_EV_FLAG FIXED BINARY(31) GLOBALDEF VALUE INIT(16), /* Display event flag */
1235 1 DISP_EV_FLAG_M BIT(32) ALIGNED GLOBALDEF VALUE INIT('0000000000000001'B), /* Display event flag mask */
1236 1 REFR_EV_FLAG FIXED BINARY(31) GLOBALDEF VALUE INIT(17), /* Refresh event flag */
1237 1 REFR_EV_FLAG_M BIT(32) ALIGNED GLOBALDEF VALUE INIT('0000000000000001'B), /* Refresh event flag mask */
1238 1 COLL_EV_FLAG FIXED BINARY(31) GLOBALDEF VALUE INIT(18), /* Collection event flag */
1239 1 BET_EV_FLAG FIXED BINARY(31) GLOBALDEF VALUE INIT(19), /* 'Between screens' event flag */
1240 1 HIB_EV_FLAG FIXED BINARY(31) GLOBALDEF VALUE INIT(20), /* Hibernation event flag */
1241 1 INTERVAL_DEFAULT FIXED BINARY(31) GLOBALDEF VALUE INIT(3), /* Default collection interval value */
1242 1 ALLCL_INT_DEFAULT FIXED BINARY(31) GLOBALDEF VALUE INIT(6), /* Default coll. interval for ALL classes Pseudo cla
1243 1 SYSCl_INT_DEFAULT FIXED BINARY(31) GLOBALDEF VALUE INIT(6), /* Default coll. interval for SYSTEM class Pseudo cl
1244 1 VIEWING_DEFAULT FIXED BINARY(31) GLOBALDEF VALUE INIT(3), /* Default viewing time value */
1245 1 FLUSH_INT_DEFAULT FIXED BINARY(31) GLOBALDEF VALUE INIT(300), /* Default flush interval value */
1246 1 BALSETMEM_DEF FIXED BINARY(31) GLOBALDEF VALUE INIT(3000), /* Default value for balance set memory */
1247 1 MPWHILIM_DEF FIXED BINARY(31) GLOBALDEF VALUE INIT(500), /* Default value for MPW_HILIMIT */
1248 1 COLLENDED BIT(1) ALIGNED GLOBALDEF, /* YES => collection has ended */
1249 1 CTRLZ_HIT BIT(1) ALIGNED GLOBALDEF, /* YES => CTRL-Z has been hit */
1250 1 CTRLC_HIT BIT(1) ALIGNED GLOBALDEF, /* YES => CTRL-C or CTRL-Z has been hit */
1251 1 NEXT_REC FIXED BINARY(31) GLOBALDEF VALUE INIT(0), /* Read next record indicator for READ INPUT rtn */
1252 1 SKIP_TO_CLASS FIXED BINARY(31) GLOBALDEF VALUE INIT(1), /* Skip to class record indicator for READ INPUT rtn
1253 1 CCDPTR POINTER GLOBALDEF, /* Pointer to CCD (Current Class Descriptor) Array */
1254 1 COLL_STATUS FIXED BINARY(31) GLOBALDEF, /* COLLECTION_EVENT status code */
1255 1 H POINTER GLOBALDEF, /* Pointer to input file header */
1256 1 INP_COMM_STR CHAR(MNR HDR$K MAXCOMLEN) GLOBALDEF, /* User comment string from input file */
1257 1 INP_COMM_LEN FIXED BINARY(15) GLOBALDEF; /* Actual length of comment string */
1258 1
1259 1
1260 1 /*
1261 1 /*
1262 1 /*
1263 1 /*
1264 1 /*
1265 1 /*
1266 1 /*
1267 1
1268 1 %REPLACE NOT_SUCCESSFUL BY '0'B; /* Failing status bit */
1269 1 %REPLACE YES BY '1'B; /* For general use */
1270 1 %REPLACE NO BY '0'B; /* For general use */
1271 1 %REPLACE ENABLE_AST BY 1; /* Enable AST indicator */
1272 1 %REPLACE DISABLE_AST BY 0; /* Disable AST indicator */
1273 1 %REPLACE AND_OP BY '0001'B; /* AND Boolean operation */
1274 1 %REPLACE XOR_OP BY '0110'B; /* XOR Boolean operation */
1275 1
```



```
1276 1  /*
1277 1  /*
1278 1  /*
1279 1  /*
1280 1  /*
1281 1  /*
1282 1  /*
1283 1  /*
1284 1  Declare
1285 1  CALL      FIXED BINARY(31),
1286 1  STATUS    BIT(1)  BASED(ADDR(CALL));
1287 1
1288 1  Declare
1289 1  NORMAL    FIXED BINARY(31) GLOBALREF,
1290 1  TEMP      FIXED BINARY(31),
1291 1  CURR_ERRCODE  FIXED BINARY(31),
1292 1  REQUEST_STATUS  FIXED BINARY(31),
1293 1  ALREADY_FAILED BIT(1)  ALIGNED,
1294 1  TEMP_PTR    POINTER,
1295 1  RECORD_STR  CHAR(128) VARYING;
1296 1
1297 1
1298 1
1299 1
1300 1
1301 1
1302 1  Declare
1303 1  INTERVAL_DEL BIT(64)  ALIGNED GLOBALDEF,
1304 1  VIEWING_DEL  BIT(64)  ALIGNED GLOBALDEF;
1305 1
1306 1  Declare
1307 1  CURR_DCLASS  FIXED BINARY(15),
1308 1  REPT_TOP     BIT(1)  ALIGNED,
1309 1  MULT_TEMP    FIXED BINARY(31) GLOBALDEF;
1310 1
1311 1  /*
1312 1  /*      File Declarations
1313 1  /*
1314 1  /*      The recording file is read shareable. This allows other
1315 1  /*      MONITOR images to use the FLUSHED recording file as input.
1316 1  /*
1317 1  /*      When the recording file is opened for update (to write header
1318 1  /*      information), it uses a fully expanded file string. This prevents
1319 1  /*      MONITOR from updating the wrong file if a new version is created
1320 1  /*      in the directory before the recording file is opened for update.
1321 1  /*
1322 1  /*
1323 1  Declare
1324 1  INPUT_FILE  FILE RECORD INPUT,
1325 1  RECORD_FILE FILE RECORD,
1326 1  RECCT       FIXED BINARY(31) GLOBALDEF;
1327 1
1328 1  Declare
1329 1  TEMP_TYPE   BIT(8)  ALIGNED;
1330 1
1331 1  Declare
```

OWN STORAGE

/* Holds function value (return status) of called ro
/* Low-order status bit for called routines */

/* MONITOR normal return status */
/* Scratch "register" */
/* MONITOR error status code currently expected */
/* EXECUTE_REQUEST status code */
/* YES => a failure has already been signaled */
/* Scratch pointer */
/* Fully expanded file name string for the recording
/* NOTE -- When the recording file is re-opened for
/* it uses this fully expanded file string. This pr
/* MONITOR from updating the wrong file if a new ver
/* created in the directory before the recording fil
/* opened for update. */

/* Delta time value for Interval */
/* Delta time value for Viewing time */

/* Consec no (not class no) of current display class
/* YES => Repeat a TOP display */
/* Temp area for MCASL_INT_MULT, used in COLLECTION_

16-SEP-1984 02:15:18 VAX-11 PL/I X2.1-273 Page 11
5-SEP-1984 15:10:53 DISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PLI;1 (8)

```

1332 1      INPUT_CPTR      POINTER GLOBALDEF,          /* Ptr to input buffer count word */
1333 1      INPUT_DATA      CHAR(MAX_REC_SIZE) VARYING BASED(INPUT_CPTR), /* Playback file input buffer */
1334 1      1 DPUT_FLAGS,    /* DISPLAY PUT routine flags */
1335 1      2 FAOL_REQUESTED BIT(8) ALIGNED,          /* YES => Xlate buffer with FAOL first */
1336 1      2 OUTP_REQUESTED BIT(8) ALIGNED,          /* YES => Really output buffer */
1337 1      PUT_LEN         FIXED BINARY(31);          /* Length of buffer for DISPLAY_PUT to put */
1338 1
1339 1

```

EXE
V04

```
1340 1 Declare
1341 1 FLUSH_IND BIT(1) ALIGNED GLOBALDEF, /* Flush indicator; YES=> perform FLUSH */
1342 1 FLUSH_COLLIS FIXED BINARY(15) GLOBALDEF, /* Number of collection events between FLUSH's */
1343 1 FLUSH_CTR FIXED BINARY(15) GLOBALDEF; /* Down counter for FLUSH_COLLIS */
1344 1
1345 1 Declare
1346 1 01 CURR_CLASS_DESCR (MAX_CLASS_NO+1), /* Current Class Descriptor Array */
1347 1 /* This array of structures includes a CCD (Current
1348 1 /* Class Descriptor) for each class collected. */
1349 1 02 CURR_CDBPTR POINTER, /* CDBPTR for current class */
1350 1 02 CURR_CLASS_NO FIXED BINARY(7); /* Class number for current class */
1351 1
1352 1
1353 1 Declare
1354 1 01 D_CURR_CLASS_DESCR (MAX_CLASS_NO+1), /* Current Class Descriptor Array for display classe
1355 1 /* This array of structures includes a D_CCD (Curren
1356 1 /* Class Descriptor) for each class displayed. */
1357 1 02 D_CURR_CDBPTR POINTER, /* CDBPTR for current display class */
1358 1 02 D_CURR_CLASS_NO FIXED BINARY(7); /* Class number for current display class */
1359 1
```



```
1360 | 1 | /*  
1361 | 1 | /*++  
1362 | 1 | /*  
1363 | 1 | /* FUNCTIONAL DESCRIPTION:  
1364 | 1 | /*  
1365 | 1 | /* EXECUTE_REQUEST  
1366 | 1 | /*  
1367 | 1 | /* This routine is called by MONMAIN to execute a MONITOR request.  
1368 | 1 | /* The request is defined by the Monitor Request Block (MRB), which  
1369 | 1 | /* is created by MONMAIN after parsing a command string.  
1370 | 1 | /*  
1371 | 1 | /* INPUTS:  
1372 | 1 | /*  
1373 | 1 | /* None  
1374 | 1 | /*  
1375 | 1 | /* IMPLICIT INPUTS:  
1376 | 1 | /*  
1377 | 1 | /* Monitor Request Block (MRB), pointed to by MRBPTR.  
1378 | 1 | /*  
1379 | 1 | /* OUTPUTS:  
1380 | 1 | /*  
1381 | 1 | /* None  
1382 | 1 | /*  
1383 | 1 | /* IMPLICIT OUTPUTS:  
1384 | 1 | /*  
1385 | 1 | /* Monitor request has been performed.  
1386 | 1 | /*  
1387 | 1 | /* ROUTINE VALUE:  
1388 | 1 | /*  
1389 | 1 | /* SSS_NORMAL, or failing MONITOR status code.  
1390 | 1 | /*  
1391 | 1 | /*--  
1392 | 1 | /*/  
1393 | 1 |
```

```
1394 1 ON FINISH; /* On finish, do nothing */
1395 1 ALREADY_FAILED = NO; /* Indicate no failure yet signaled */
1396 1 CURR_ERRCODE = 0; /* Set expected MONITOR code to default */
1397 1
1398 1 /*
1399 1 /* Set up condition handler to terminate the MONITOR request on:
1400 1 /* 1) any asynchronous error condition, such as file and I/O errors;
1401 1 /* 2) any synchronous MONITOR-detected condition.
1402 1 /*
1403 1
1404 1 ON ANYCONDITION /* On any condition signaled, */
1405 1 BEGIN;
1406 2
1407 2 Declare
1408 2 MNR$_ERRINPFIL FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
1409 2 MNR$_ERRRECFIL FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
1410 2 MNR$_UNEXPERR FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
1411 2 MON_CODE FIXED BINARY(31), /* Monitor message code */
1412 2 TEMP FIXED BINARY(31), /* Temporary scratch area */
1413 2 MNR$_FACNO FIXED BINARY(31) GLOBALREF VALUE, /* MONITOR facility number */
1414 2 ON_FILE CHAR(100) VARYING, /* Holds possible file name string */
1415 2 SIGNALLED_ERR ENTRY (ANY VALUE, ANY VALUE, ANY VALUE, ANY); /* Rtn to set up PUTMSGVEC */
1416 2
1417 2 IF ^ ALREADY_FAILED /* If a failure not already signaled, */
1418 2 THEN DO;
1419 2 ALREADY_FAILED = YES; /* Indicate a failure has been signaled */
1420 2 CHF$ARGPTR = ONARGSLIST(); /* Get signal array pointer */
1421 2 ST$VALUE = CHF$SIG_NAME; /* Get code for signaled condition */
1422 2 UNSPEC(TEMP) = ST$FAC_NO; /* Convert facility no. to binary in TEMP */
1423 2 IF TEMP = MNR$_FACNO /* If a MONITOR code, */
1424 2 THEN MON_CODE = ST$VALUE; /* then remember it */
1425 2 ELSE DO; /* Otherwise, need to set the MON_CODE */
1426 2 ON_FILE = ONFILE(); /* Get PL/I file constant if I/O cond */
1427 2 IF ON_FILE = 'INPUT_FILE' /* If input file error, */
1428 2 THEN MON_CODE = MNR$_ERRINPFIL; /* Set Monitor status code accordingly */
1429 2 ELSE IF ON_FILE = 'RECORD_FILE' /* See if it's the recording file */
1430 2 THEN MON_CODE = MNR$_ERRRECFIL; /* Yes, save code */
1431 2 ELSE IF CURR_ERRCODE = 0 /* No, see if an error is currently expected */
1432 2 THEN MON_CODE = MNR$_UNEXPERR; /* No, set 'unexpected' code */
1433 2 ELSE MON_CODE = CURR_ERRCODE; /* Yes, set currently expected code */
1434 2 CURR_ERRCODE = 0; /* Reset to default MONITOR error code ('une
1435 2 CALL SIGNALLED_ERR(MON_CODE, ST$VALUE, DIM(CHF$SIG_ARG, 1), CHF$SIG_ARG); /* Log the error */
1436 2 END;
1437 2
1438 2 REQUEST_STATUS = MON_CODE; /* Set up code for MONITOR request termination */
1439 2 CALL COLLECTION_END(); /* Shut down collection activity */
1440 2 CALL REQUEST_CLEANUP(); /* Perform cleanup for files, memory, etc. */
1441 2 END;
1442 2
1443 2 GO TO REQUEST_EXIT; /* Go return from EXECUTE_REQUEST (PL/I does an UNWI
1444 2
1445 2 END; /* End of ON-condition routine */
1446 1
```



```
1447 : 1 /*
1448 : 1 /*      Set up EOF condition
1449 : 1 /*/
1450 : 1
1451 : 1 IF M->MRBSA INPUT ^= NULL() /* If this is a PLAYBACK request, */
1452 : 1 THEN ON ENDFILE(INPUT_FILE) MC->MCA$V_EOF = YES; /* then set up EOF condition */
1453 : 1
1454 : 1 /*
1455 : 1 /*      General MONITOR request initialization
1456 : 1 /*/
1457 : 1
1458 : 1 CALL = REQUEST_INIT(); /* Initialization for this request */
1459 : 1 IF STATUS = NOT_SUCCESSFUL
1460 : 1 THEN CALL SIGNAL_MON_ERR(); /* Short-circuit request if failure */
1461 : 1
1462 : 1 /*
1463 : 1 /*      Establish CTRL-C and CTRL-Z handlers for terminating the MONITOR request.
1464 : 1 /*      CTRL-C causes a MONITOR> prompt. CTRL-Z returns to DCL.
1465 : 1 /*/
1466 : 1
1467 : 1 IF COLLENDED = NO THEN CALL = ESTAB_CTRL CZ(); /* If still collecting, establish CTRL-C and CTRL-Z handlers
1468 : 1 /* If error, do not terminate; simply ignore CTRL-C's and CT
1469 : 1
1470 : 1 /*
1471 : 1 /*      Establish CTRL-W handler for refreshing the terminal display.
1472 : 1 /*/
1473 : 1
1474 : 1 IF COLLENDED = NO & M->MRBSV_DISPLAY /* If still collecting, and display requested */
1475 : 1 THEN CALL = ESTAB_CTRL W(); /* ... establish CTRL-W handler */
1476 : 1 /* If error, do not terminate; simply ignore CTRL-W's */
1477 : 1
```

```
1478 : 1 /*
1479 : 1 /* If this is a live request, beginning in the future, 'hibernate'
1480 : 1 /* the process until ready to execute. (Use event flags instead of $HIBER
1481 : 1 /* to avoid the problem of outstanding wakeups interfering with later
1482 : 1 /* MONITOR requests.)
1483 : 1 /*/
1484 : 1
1485 : 1 IF ^ M->MRBSV_PLAYBACK & MC->MCASV_FUTURE /* If live future request, */
1486 : 1 & COLLENDED = NO /* ... and not terminated, */
1487 : 1 THEN DO;
1488 : 2 CALL = SYSS$SETIMR(HIB_EV_FLAG,M->MRBSQ_BEGINNING,,HIB_EV_FLAG);
1489 : 2 /* ... set flag when ready to execute request */
1490 : 2 IF STATUS = NOT_SUCCESSFUL /* Failed? */
1491 : 2 THEN DO;
1492 : 3 CALL MON_ERR(MNR$ SSERROR,CALL,SETIMR_STR); /* Yes -- log the error */
1493 : 3 CALL SIGNAL_MON_ERR(); /* ... and signal it */
1494 : 3 END;
1495 : 2 BEGIN;
1496 : 2 DECLARE 1 HIBMSG, /* Declare hibernate msg vec dynamically */
1497 : 2 HCOUNT FIXED BIN(31) INIT(1),
1498 : 2 HMSG FIXED BIN(31) INIT(MNR$_HIB);
1499 : 2 CALL = SYSS$PUTMSG(HIBMSG,,); /* Let user know we're sleeping */
1500 : 2 END;
1501 : 2 IF COLLENDED = NO THEN CALL = SYSS$WAITFR(HIB_EV_FLAG); /* ... ZZZZZZZZZZ */
1502 : 2 END;
1503 : 1
1504 : 1 /*
1505 : 1 /* Initialization associated with /DISPLAY output
1506 : 1 /*/
1507 : 1
1508 : 1 IF M->MRBSV_DISPLAY /* If DISPLAY has been requested, */
1509 : 1 & COLLENDED = NO /* ... and request not terminated, */
1510 : 1 THEN DO;
1511 : 2 CALL = DISPLAY_INIT(); /* ... then perform init for it */
1512 : 2 IF STATUS = NOT_SUCCESSFUL /* Failed? */
1513 : 2 THEN DO;
1514 : 3 CALL MON_ERR(MNR$ DISPERR,CALL); /* Yes -- log the error */
1515 : 3 CALL SIGNAL_MON_ERR(); /* ... and signal it */
1516 : 3 END;
1517 : 2 ON FINISH CALL = DISPLAY_CLEANUP(); /* On finish, do display cleanup */
1518 : 2 END;
1519 : 1
1520 : 1 /*
1521 : 1 /* Initialization associated with /RECORD output
1522 : 1 /*/
1523 : 1
1524 : 1 IF M->MRBSV_RECORD /* If RECORD has been requested, */
1525 : 1 & COLLENDED = NO /* ... and request not terminated, */
1526 : 1 THEN DO;
1527 : 2 CALL = RECORD_INIT(); /* ... then do init for it */
1528 : 2 IF STATUS = NOT_SUCCESSFUL /* Signal error if failure */
1529 : 2 THEN CALL SIGNAL_MON_ERR();
1530 : 2 END;
1531 : 1
```



```
1532 | 1 /*  
1533 | 1 /* Execute main monitoring routine. When control returns from the CALL,  
1534 | 1 /* the MONITOR request will have terminated.  
1535 | 1 /*/  
1536 | 1  
1537 | 1 IF COLLENDED = NO /* If collection not ended, */  
1538 | 1 THEN DO;  
1539 | 2 CALL = MONITOR_REQUEST(); /* Perform the MONITOR request */  
1540 | 2 IF STATUS = NOT_SUCCESSFUL /* If failed, */  
1541 | 2 THEN CALL SIGNAL_MON_ERR(); /* ... signal the error */  
1542 | 2 END;  
1543 | 1  
1544 | 1 /*  
1545 | 1 /* Perform SUMMARY processing if requested.  
1546 | 1 /*/  
1547 | 1  
1548 | 1 IF M->MRBSV_SUMMARY /* If SUMMARY has been requested, */  
1549 | 1 THEN DO;  
1550 | 2 CALL = REQUEST_SUMMARY(); /* Perform SUMMARY processing */  
1551 | 2 IF STATUS = NOT_SUCCESSFUL /* If failed, */  
1552 | 2 THEN CALL SIGNAL_MON_ERR(); /* ... signal the error */  
1553 | 2 END;  
1554 | 1  
1555 | 1 /*  
1556 | 1 /* Perform Multi-File Summary processing if requested.  
1557 | 1 /*/  
1558 | 1  
1559 | 1 IF M->MRBSV_MFSUM /* If Multi-File Summary has been requested, */  
1560 | 1 THEN DO;  
1561 | 2 CALL = SAVE_SUM_BUFFS(); /* Save SUM buffers */  
1562 | 2 IF STATUS = NOT_SUCCESSFUL /* Failed? */  
1563 | 2 THEN DO;  
1564 | 3 CALL MON_ERR(MNRS_UNEXPERR,CALL); /* Yes -- log the error */  
1565 | 3 CALL SIGNAL_MON_ERR(); /* ... and signal it */  
1566 | 3 END;  
1567 | 2 END;  
1568 | 1  
1569 | 1 /*  
1570 | 1 /* Cleanup processing  
1571 | 1 /*/  
1572 | 1  
1573 | 1 CALL REQUEST_CLEANUP(); /* Execute various cleanup routines */  
1574 | 1  
1575 | 1 /*  
1576 | 1 /* Exit from EXECUTE_REQUEST routine  
1577 | 1 /* Note -- we get to this point either by falling through  
1578 | 1 /* the above code (normal path), or by direct branch from  
1579 | 1 /* the condition-handling routine (error path).  
1580 | 1 /*/  
1581 | 1  
1582 | 1 REQUEST_STATUS = NORMAL; /* Normal status if we get to this point */  
1583 | 1  
1584 | 1 REQUEST_EXIT:  
1585 | 1  
1586 | 1 RETURN(REQUEST_STATUS); /* Return to MONMAIN.PLI with status */  
1587 | 1
```

```
1588 1 REQUEST_INIT: Procedure Returns(fixed binary(31));
1589 2
1590 3 /*
1591 4 /*++
1592 5 /*
1593 6 /* FUNCTIONAL DESCRIPTION:
1594 7 /*
1595 8 /* REQUEST_INIT
1596 9 /*
1597 10 /* Performs initialization for the Monitor request.
1598 11 /* Fills in defaults for the MRB (Monitor Request Block).
1599 12 /* Also inits the MCA (Monitor Communication Area), opens
1600 13 /* the input (playback) file if necessary, and fills in the
1601 14 /* SYI (System Information Area).
1602 15 /*
1603 16 /* INPUTS:
1604 17 /*
1605 18 /* None
1606 19 /*
1607 20 /* OUTPUTS:
1608 21 /*
1609 22 /* None
1610 23 /*
1611 24 /* ROUTINE VALUE:
1612 25 /*
1613 26 /* SSS_NORMAL, or failing MONITOR status code.
1614 27 /*
1615 28 /* SIDE EFFECTS:
1616 29 /*
1617 30 /* /INPUT file (INPUT_FILE) is positioned to the first class record.
1618 31 /*
1619 32 /*--
1620 33 /*/
1621 34
1622 35 /*
1623 36 /*
1624 37 /*
1625 38 /* LOCAL STORAGE
1626 39 /*
1627 40 /*
1628 41 /*/
1629 42
1630 43 Declare
1631 44 F POINTER; /* Pointer to class record in input file */
1632 45
```



```
1633      REQUEST STATUS = NORMAL;
1634      COLL STATUS = NORMAL;
1635      COLLENDED = NO;
1636      CTRLZ_HIT = NO;
1637      CTRLCZ_HIT = NO;
1638      CTRLCZ_CHAN = 0;
1639      CTRLW_CHAN = 0;
1640      INP_COMM_LEN = 0;
1641      MC->MCASL_COLLCNT = 0;
1642      MC->MCASL_DISPCNT = 0;
1643      MC->MCASL_CONSEC_REC = 0;
1644      MC->MCASL_LASTCOLL = '0'B;
1645      MC->MCASV_FUTURE = NO;
1646      MC->MCASV_EOF = NO;
1647      MC->MCASV_ERA_SCRL = NO;
1648      MC->MCASV_VIDEO = NO;
1649      MC->MCASV_GRAPHICS = NO;
1650      MC->MCASV_TOP_DISP = NO;
1651      MC->MCASV_S_TOP_DISP = NO;
1652      MC->MCASV_REFRESH = NO;
1653      MC->MCASV_FILLER = '0'B;
1654      FLUSH_IND = NO;
1655      CURR_BCLASS = 0;
1656      REPT_TOP = NO;
1657      DISPLAYING = NO;
1658      CCDPTR = ADDR(CURR_CLASS_DESCR);
1659      CALL = SYSSCLREF(REFR_EV_FLAG);
1660      CALL = SYSSSETAST(ENABLE_AST);
1661
1662      /*
1663      /*      Set MRB flags for options that were requested
1664      /*/
1665
1666      IF M->MRBSA_INPUT ^= NULL() THEN M->MRBSV_PLAYBACK = YES; /* If INPUT specified, indicate so */
1667      IF M->MRBSA_DISPLAY ^= NULL() THEN M->MRBSV_DISPLAY = YES; /* If DISPLAY specified, indicate so */
1668      IF M->MRBSA_RECORD ^= NULL() THEN M->MRBSV_RECORD = YES; /* If RECORD specified, indicate so */
1669      IF M->MRBSA_SUMMARY ^= NULL() THEN M->MRBSV_SUMMARY = YES; /* If SUMMARY specified, indicate so */
1670
1671      IF ^ M->MRBSV_DISPLAY & ^ M->MRBSV_RECORD & ^ M->MRBSV_SUMMARY /* If none of the outputs requested, */
1672      & ^ M->MRBSV_MFSUM /* ... AND it's not a m.f. summary, */
1673      THEN DO;
1674          CALL MON_ERR(MNRS_NOOUTPUT); /* Log the error */
1675          RETURN(MNRS_NOOUTPUT); /* ... and return with status */
1676      END;
1677
1678      /*
1679      /*      Set or clear display event flag
1680      /*/
1681
1682      IF M->MRBSV_DISPLAY /* If display requested, */
1683      THEN CALL = SYSSSETEF(DISP_EV_FLAG); /* ... then set display event flag to force 1st display even
1684      ELSE CALL = SYSSCLREF(DISP_EV_FLAG); /* ... otherwise clear it */
1685
1686      /*
1687      /*      If PLAYBACK, perform input file initialization, so MONITOR file header information can be accessed.
1688      /*/
```

EXECUTE_REQUEST
V04-000

K 7
16-SEP-1984 02:15:22
5-SEP-1984 15:10:53

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PLI;1 (16)

Page 20

1689
1690
1691
1692
1693
1694
1695

2
2
2
2
2
2
2

IF M->MRBSV_PLAYBACK

THEN DO;

CALL = INPUT_INIT();

IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);

END;

/* If this is a PLAYBACK request, */

/* ... perform input file initialization */


```
1696      IF M->MRBSV_PLAYBACK          /* If this is a PLAYBACK request, */
1697      THEN DO:
1698          INP_COMM_STR = H->MNR_HDRST_COMMENT;    /* Save user's comment string from header record */
1699          INP_COMM_LEN = H->MNR_HDRSW_COMLEN;      /* ... and its length */
1700      END;
1701
1702      /*
1703      /* The next several groups of code update the MRB with default and
1704      /* specified values, and, for PLAYBACK, values from the input file.
1705      /*
1706
1707      /*
1708      /* Verify requested classes and set up Current Class Descriptor array
1709      /*
1710
1711      BEGIN;
1712      Declare
1713      SELECT_REV_LEVS ENTRY(BIT(128) ALIGNED, BIT(128) ALIGNED, CHAR(128), ANY) /* MACRO-32 rtn to select revision levels
1714
1715      CALC_LEN          OPTIONS(VARIABLE),          /* ... for all classes */
1716                      ENTRY(BIT(128) ALIGNED)      /* MACRO-32 rtn to calculate class block (
1717                      RETURNS (FIXED BINARY(31));
1718      Declare
1719      REVLEVELS         (0:127) FIXED BINARY(7) GLOBALDEF,          /* Revision Levels Vector */
1720      REVOCLSBITS       BIT(128) GLOBALDEF,                        /* Monitored classes still at Rev Level 0 */
1721      REVOCB_VEC        (0:127) BIT(1) DEFINED(REVOCLSBITS);        /* Bit-addressable alias */
1722      Declare
1723      FILE_CLASSES      BIT(128),          /* Classes from input file */
1724      REQ_CLASSES       BIT(128),          /* Classes requested by user */
1725      NP_CLASSES        BIT(MAX_CLASS_NO+1), /* Classes requested but not in input file */
1726      DO_CLASSES        BIT(128),          /* Classes to actually monitor */
1727      DO_CLASSES_VEC    (0:127) BIT(1) DEFINED(DO_CLASSES),        /* Bit-addressable alias */
1728      DO_CLASSES_AL     BIT(128) ALIGNED,  /* Aligned copy of DO_CLASSES */
1729      UNK_CLASSES       BIT(128) ALIGNED,  /* Classes with unknown revision levels */
1730      DISPLAY_CLASSES   BIT(128) ALIGNED GLOBALDEF,               /* Classes to be displayed */
1731      CDBHEAD           FIXED BINARY(31) GLOBALREF VALUE,          /* Address of first CDB */
1732      I                 FIXED BINARY(15),                          /* Index for do-loop */
1733      CLASS_NO          FIXED BINARY(7),                          /* Class number */
1734      TEMP_CDBPTR       POINTER,          /* Ptr to Class Descriptor Block (CDB) */
1735      CDBPTR_COMP       FIXED BINARY(31) BASED(ADDR(TEMP_CDBPTR)); /* Computable CDBPTR */
1736
1737      Declare
1738      SYS_REQ           BIT(1) ALIGNED,          /* YES => SYSTEM class requested */
1739      PROCS_REQ         BIT(1) ALIGNED,          /* YES => PROCESSES class requested */
1740      STATES_REQ        BIT(1) ALIGNED,          /* YES => STATES class requested */
1741      MODES_REQ         BIT(1) ALIGNED;          /* YES => MODES class requested */
```

```
1742 3 DO I = 0 TO 127;
1743 4 REVLEVELS(I) = 0;
1744 4 END;
1745 3
1746 3 DO_CLASSES = M->MRB$O_CLASSBITS;
1747 3
1748 3 IF M->MRB$V_PLAYBACK
1749 3 THEN DO;
1750 4 IF MNR_HDR$K_CLASSBITS < MC->MCAS$ INPUT_LEN
1751 4 THEN FILE_CLASSES = H->MNR_HDR$O_CLASSBITS;
1752 4 ELSE FILE_CLASSES = H->MNR_HDR$O_REVOCLSBITS;
1753 4
1754 4
1755 4 REQ_CLASSES = DO_CLASSES;
1756 4 DO_CLASSES = BOOL(FILE_CLASSES, REQ_CLASSES, AND_OP);
1757 4 NP_CLASSES = BOOL(DO_CLASSES, REQ_CLASSES, XOR_OP);
1758 4 IF DO_CLASSES = '0'B
1759 4 THEN DO;
1760 5 CALL MON_ERR(MNR$ NOCLASS);
1761 5 RETURN(MNR$ NOCLASS);
1762 5 END;
1763 4
1764 4 IF M->MRB$V_DISPLAY = NO & NP_CLASSES ^= '0'B
1765 4 & M->MRB$V_MFSUM = NO & M->MRB$V_ALL_CLASS = NO
1766 4 THEN BEGIN;
1767 5 DECLARE 1 NPMSG,
1768 5 2 NPCOUNT FIXED BIN(31) INIT(1),
1769 5 2 NPMSG FIXED BIN(31) INIT(MNR$ CLASNP);
1770 5 CALL = SYS$PUTMSG(NPMSG,,);
1771 5 END;
1772 4
1773 3
```

/* Set all revision levels ... */
/* ... to 0 */

/* Get set of classes to do */

/* Playback request */

/* If CLASSBITS field is defined for input file, */
/* then get file classes from usual place */
/* else get them from another place */
/* NOTE -- MNR_HDR\$O_REVOCLSBITS is used for compati
/* with MONS001 and MONBA001 file struct le

/* Get requested classes */
/* Compute classes to actually do */
/* Compute classes not present */
/* If no classes to be done, */

/* Log error */
/* ... and return */

/* If at least one class not present AND not display
/* ... AND not multi-file summary, AND not ALL_CLASS
/* ... then print a warning */
/* Declare not present msg vec dynamically */

/* Warn user that classes missing */


```
1774 3 IF DO_CLASSES_VEC(SYSTEM_CLSNO) /* If SYSTEM class requested, */
1775 3 THEN DO;
1776 4 SYS_REQ = YES; /* Remember that fact */
1777 4 PROCS_REQ = DO_CLASSES_VEC(PROCS_CLSNO); /* Remember whether PROCESSES requested */
1778 4 DO_CLASSES_VEC(PROCS_CLSNO) = YES; /* ... and include it */
1779 4 STATES_REQ = DO_CLASSES_VEC(STATES_CLSNO); /* Remember whether STATES requested */
1780 4 DO_CLASSES_VEC(STATES_CLSNO) = YES; /* ... and include it */
1781 4 MODES_REQ = DO_CLASSES_VEC(MODES_CLSNO); /* Remember whether MODES requested */
1782 4 DO_CLASSES_VEC(MODES_CLSNO) = YES; /* ... and include it */
1783 4 END;
1784 3 ELSE SYS_REQ = NO; /* Indicate SYSTEM class not requested */
1785 3
1786 3 DO_CLASSES_AL = DO_CLASSES; /* Get aligned string for later routine calls */
1787 3
1788 3 IF M->MRB$V_PLAYBACK /* Playback request */
1789 3 THEN DO;
1790 4 /*
1791 4 /* For each class in DO_CLASSES, update the CDB with information
1792 4 /* from the CHD (CHange-Descriptor) for the appropriate revision
1793 4 /* level.
1794 4 /*
1795 4 /* Eliminate from DO_CLASSES those classes with incompatible structure
1796 4 /* levels. Issue a warning message if any incompatibilities found.
1797 4 /*/
1798 4
1799 4 IF MNR_HDR$K_REVLEVELS < MC->MCASL_INPUT_LEN /* If REVLEVELS field is defined for input file, */
1800 4 THEN DO;
1801 5 CALL SELECT_REV_LEVS(DO_CLASSES_AL, UNK_CLASSES, /* Select revision levels ... */
1802 5 H->MNR_HDR$T_REVLEVELS, REVLEVELS); /* ... for all classes */
1803 5
1804 5 IF UNK_CLASSES ^= '0'B /* If at least one class has unknown rev level, */
1805 5 THEN DO; /*
1806 6 DO_CLASSES = BOOL(DO_CLASSES, UNK_CLASSES, XOR_OP); /* Remove unknowns from DO_CLASSES */
1807 6 IF M->MRB$V_DISPLAY = NO /* If not displaying, */
1808 6 THEN BEGIN; /* ... then print a warning */
1809 7 DECLARE 1 UNKMSG, /* Declare unknown msg vec dynamically */
1810 7 2 UNKCOUNT FIXED BIN(31) INIT(1),
1811 7 2 UNKMSG FIXED BIN(31) INIT(MNR$CLASUNK);
1812 7 CALL = SYSPUTMSG(UNKMSG,); /* Warn user that classes have unknown structs */
1813 7 END;
1814 6 END;
1815 5 END;
1816 4
1817 4 ELSE /* Revision levels all 0 */
1818 4 CALL SELECT_REV_LEVS(DO_CLASSES_AL, REVLEVELS); /* Move CHD info into CDB */
1819 4 /* ... for all classes */
1820 4 END;
1821 3
1822 3 ELSE DO; /* Live request */
1823 4 CALL SELECT_REV_LEVS(DO_CLASSES_AL, REVLEVELS); /* Select revision levels for all classes */
1824 4 END;
1825 3
1826 3 IF DO_CLASSES_VEC(SYSTEM_CLSNO) = YES /* If SYSTEM class being monitored, */
1827 3 THEN M->MRB$V_SYSCLS = YES; /* then indicate so */
1828 3 ELSE DO;
1829 4 M->MRB$V_SYSCLS = NO; /* else indicate not */
```



```
1830 4      IF SYS_REQ = YES
1831 4      THEN DO;
1832 3          IF PROCS_REQ = NO
1833 3              THEN DO_CLASSES_VEC(PROCS_CLSNO) = NO;
1834 3          IF STATES_REQ = NO
1835 3              THEN DO_CLASSES_VEC(STATES_CLSNO) = NO;
1836 3          IF MODES_REQ = NO
1837 3              THEN DO_CLASSES_VEC(MODES_CLSNO) = NO;
1838 3          END;
1839 4      END;
1840 3
1841 3      IF M->MRB$V_MFSUM
1842 3          THEN DO_CLASSES_VEC(PROCS_CLSNO) = NO;
1843 3
1844 3      IF DO_CLASSES = '0'B
1845 3          THEN DO;
1846 4          CALL MON_ERR(MNR$_NOCLASS);
1847 4          RETURN(MNR$_NOCLASS);
1848 4      END;
1849 3
1850 3      DISPLAY_CLASSES = M->MRB$O_CLASSBITS;
1851 3      DISPLAY_CLASSES = BOOL(DISPLAY_CLASSES,DO_CLASSES,AND_OP);
1852 3      M->MRB$O_CLASSBITS = DO_CLASSES;
1853 3
```

/* If SYSTEM originally requested, */
/* then make some further checks */
/* If PROCESSES not originally requested, */
/* ... make sure it's off now */
/* If STATES not originally requested, */
/* ... make sure it's off now */
/* If MODES not originally requested, */
/* ... make sure it's off now */

/* If multi-file summary, */
/* then make sure PROCESSES class is still off */

/* If no classes to be done, */

/* Log error */
/* ... and return */

/* Get unaligned copy of orig requested classes */
/* Compute classes to display */
/* Remember classes to collect */


```
1854 : /*
1855 : /*      Given DO_CLASSES, execute do loop using INDEX builtin
1856 : /*      to fill in the CCD (Current Class Descriptor) array.
1857 : /*      When do loop is finished, MRBSO_CLASSBITS, MRBSW_CLASSCT
1858 : /*      MCASB_FIRSTC and MCASB_LASTC
1859 : /*      will all be established.
1860 : /*/
1861 :
1862 : REVOCLSBITS = '0'B;
1863 : CALL = CALC_LEN(M->MRBSO_CLASSBITS);
1864 : IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);
1865 :
1866 : CLASS_NO = 0;
1867 : DO I = 1 TO MAX_CLASS_NO + 1 WHILE (CLASS_NO >= 0);
1868 :   CLASS_NO = INDEX(DO_CLASSES, YES) - 1;
1869 :   IF CLASS_NO >= 0
1870 :     THEN DO;
1871 :       DO_CLASSES_VEC(CLASS_NO) = NO;
1872 :       IF REVLEVELS(CLASS_NO) = 0
1873 :         THEN REVOCB_VEC(CLASS_NO) = YES;
1874 :       CURR_CLASS_NO(I) = CLASS_NO;
1875 :       IF I = 1 THEN MC->MCASB_FIRSTC = CLASS_NO;
1876 :       MC->MCASB_LASTC = CLASS_NO;
1877 :       M->MRBSW_CLASSCT = I;
1878 :       CDBPTR_COMP = CDBHEAD + (CDB$K_SIZE * CURR_CLASS_NO(I));
1879 :       CURR_CDBPTR(I) = TEMP_CDBPTR;
1880 :     END;
1881 :   END;
1882 :
1883 : /*
1884 : /*      Now, given DISPLAY_CLASSES, do a similar loop as above to set up D_CCD,
1885 : /*      the display version of the CCD. When loop is finished, the array will
1886 : /*      be established along with MCASW_DCLASSCT, the number of display classes.
1887 : /*/
1888 :
1889 : DO_CLASSES = DISPLAY_CLASSES;
1890 : IF DO_CLASSES_VEC(PROCS_CLSNO)
1891 :   THEN M->MRBSV_PROC_REQ = YES;
1892 : ELSE M->MRBSV_PROC_REQ = NO;
1893 : CLASS_NO = 0;
1894 : DO I = 1 TO MAX_CLASS_NO + 1 WHILE (CLASS_NO >= 0);
1895 :   CLASS_NO = INDEX(DO_CLASSES, YES) - 1;
1896 :   IF CLASS_NO >= 0
1897 :     THEN DO;
1898 :       DO_CLASSES_VEC(CLASS_NO) = NO;
1899 :       D_CURR_CLASS_NO(I) = CLASS_NO;
1900 :       MC->MCASW_DCLASSCT = I;
1901 :       CDBPTR_COMP = CDBHEAD + (CDB$K_SIZE * D_CURR_CLASS_NO(I));
1902 :       D_CURR_CDBPTR(I) = TEMP_CDBPTR;
1903 :     END;
1904 :   END;
1905 :
1906 : END;
1907 :
1908 : END;
1909 :
```

/* Assume no classes at Rev Level 0 */
/* Calc CDB\$W_BLKLEN field for each class */
/* Return if error */

/* Initialize class number */
/* Loop once for each possible class */
/* Find next requested class number */
/* Only continue if a class was found */

/* Eliminate it from future consideration */
/* If this class is at Rev Level 0, */
/* then indicate so */
/* Store class_no in CCD table */
/* Mark first class requested */
/* Mark last class requested */
/* Keep track of class count */

/* Compute current cdbptr ... */
/* ... and store it in CCD table */

/* Use DO_CLASSES vector */
/* If PROCESSES to be displayed, */
/* then indicate it was requested */
/* else indicate not */
/* Initialize class number */
/* Loop once for each possible class */
/* Find next requested class number */
/* Only continue if a class was found */

/* Eliminate it from future consideration */
/* Store class_no in D_CCD table */
/* Keep track of class count */

/* Compute current cdbptr ... */
/* ... and store it in D_CCD table */

/* End of BEGIN-END group */

```
1910 : 2 /*
1911 : 2 /* Establish defaults for FLUSH_INTERVAL, INTERVAL and VIEWING_TIME
1912 : 2 /* options.
1913 : 2 /* If Playback, divide file value for INTERVAL into requested value, and round
1914 : 2 /* requested value up to the next whole multiple of the file value. Store the
1915 : 2 /* multiple value in MCASL_INT_MULT. It will be used to trigger recording and
1916 : 2 /* display events.
1917 : 2 /*/
1918 : 2
1919 : 2 IF M->MRBSL_FLUSH = 0 /* If FLUSH never specified... */
1920 : 2 THEN M->MRBSL_FLUSH = FLUSH_INT_DEFAULT; /* normal default value */
1921 : 2
1922 : 2 IF M->MRBSV_PLAYBACK /* Playback request */
1923 : 2 THEN DO:
1924 : 2 IF M->MRBSL_VIEWING_TIME = 0 /* If VIEWING_TIME never specified, */
1925 : 2 THEN M->MRBSL_VIEWING_TIME = VIEWING_DEFAULT; /* ... then take default */
1926 : 2 IF M->MRBSL_INTERVAL = 0 /* If INTERVAL never specified, */
1927 : 2 THEN DO:
1928 : 2 M->MRBSL_INTERVAL = H->MNR_HDRSL_INTERVAL; /* ... then use file value */
1929 : 2 MC->MCASL_INT_MULT = 1; /* ... and multiple of 1 */
1930 : 2 END;
1931 : 2 ELSE DO: /* INTERVAL explicitly specified */
1932 : 2 MC->MCASL_INT_MULT = DIVIDE(M->MRBSL_INTERVAL, H->MNR_HDRSL_INTERVAL, 31); /* Divide spec'd val by file val */
1933 : 2 IF (M->MRBSL_INTERVAL - (H->MNR_HDRSL_INTERVAL * MC->MCASL_INT_MULT)) ^= 0 /* Divide spec'd val by file val */
1934 : 2 THEN DO:
1935 : 2 MC->MCASL_INT_MULT = MC->MCASL_INT_MULT + 1; /* Round up if necessary */
1936 : 2 M->MRBSL_INTERVAL = MC->MCASL_INT_MULT * H->MNR_HDRSL_INTERVAL; /* Round interval too */
1937 : 2 END;
1938 : 2 END;
1939 : 2
1940 : 2 END;
1941 : 2
1942 : 2 ELSE DO: /* Live request */
1943 : 2 IF M->MRBSL_INTERVAL = 0 /* If INTERVAL never specified... */
1944 : 2 THEN IF M->MRBSV_ALL_CLASS /* ALL class request */
1945 : 2 THEN M->MRBSL_INTERVAL = ALLCL_INT_DEFAULT; /* ALL class default value */
1946 : 2 ELSE IF M->MRBSV_SYSCLS /* SYSTEM class request */
1947 : 2 THEN M->MRBSL_INTERVAL = SYSC_INT_DEFAULT; /* SYSTEM class default value */
1948 : 2 ELSE M->MRBSL_INTERVAL = INTERVAL_DEFAULT; /* normal default value */
1949 : 2 IF M->MRBSL_VIEWING_TIME = 0 /* If VIEWING_TIME never specified... */
1950 : 2 THEN M->MRBSL_VIEWING_TIME = M->MRBSL_INTERVAL; /* Default to interval value */
1951 : 2 IF M->MRBSL_INTERVAL <= M->MRBSL_FLUSH /* Requested interval not larger than flush period? */
1952 : 2 THEN FLOSH_COLL = DIVIDE(M->MRBSL_FLUSH, M->MRBSL_INTERVAL, 31); /* Yes -- compute collections until flu */
1953 : 2 ELSE FLOSH_COLL = 1; /* No -- flush on every collection */
1954 : 2 FLOSH_CTR = FLOSH_COLL; /* Set up down counter */
1955 : 2
1956 : 2 END;
1957 : 2
1958 : 2 CALL CVT_TO_DELTA(M->MRBSL_INTERVAL, INTERVAL_DEL); /* Convert INTERVAL to delta time */
1959 : 2 CALL CVT_TO_DELTA(M->MRBSL_VIEWING_TIME, VIEWING_DEL); /* Convert VIEWING_TIME to delta time */
1960 : 2
```



```
1961 : /*
1962 : /* Establish defaults for BEGINNING and ENDING options
1963 : /*/
1964 :
1965 : IF M->MRBSV_PLAYBACK
1966 : THEN MC->MCASQ_CURR_TIME = H->MNR_HDRSQ_BEGINNING; /* If Playback, get current time from file */
1967 : /* If Live, MCASQ_CURR_TIME already contains */
1968 : /* ... current time from system */
1969 :
1970 : /*
1971 : /* If user requested a past time for the BEGINNING option,
1972 : /* or defaulted, then replace his value with MCASQ_CURR_TIME.
1973 : /* Otherwise, indicate a future request.
1974 : /*/
1975 :
1976 : MC->MCASV_FUTURE = QUAD_LT_QUAD(MC->MCASQ_CURR_TIME,M->MRBSQ_BEGINNING); /* MCASV_FUTURE gets YES or NO */
1977 : IF MC->MCASV_FUTURE = NO
1978 : THEN M->MRBSQ_BEGINNING = MC->MCASQ_CURR_TIME; /* If NO, give user current time */
1979 :
1980 : /*
1981 : /* For PLAYBACK, verify ENDING option. If file value is
1982 : /* non-zero, replace requested value with file value if
1983 : /* requested value is 0 (never specified), or requested
1984 : /* value is larger (later) than file value.
1985 : /*/
1986 :
1987 : IF M->MRBSV_PLAYBACK
1988 : THEN IF ^ QUAD EQ 0(H->MNR_HDRSQ_ENDING)
1989 : THEN IF QUAD EQ 0(M->MRBSQ_ENDING)
1990 : THEN M->MRBSQ_ENDING = H->MNR_HDRSQ_ENDING;
1991 : ELSE IF QUAD LT QUAD(H->MNR_HDRSQ_ENDING,M->MRBSQ_ENDING)
1992 : THEN M->MRBSQ_ENDING = H->MNR_HDRSQ_ENDING;
1993 :
1994 : /*
1995 : /* Set indefinite end bit if ENDING option never specified.
1996 : /*
1997 : /* Also, perform sanity check of BEGINNING and ENDING times.
1998 : /* If BEGINNING not less than ENDING, exit with error.
1999 : /*/
2000 :
2001 : IF QUAD EQ 0(M->MRBSQ_ENDING) /* If ENDING never specified, */
2002 : THEN M->MRBSV_INDEFEND = YES; /* ... call it indefinite */
2003 : ELSE IF QUAD LT QUAD(M->MRBSQ_BEGINNING,M->MRBSQ_ENDING) = NO /* If BEGINNING not less than ENDING, */
2004 : THEN DO; /* Log the error */
2005 : CALL MON_ERR(MNRS_BEGNLEND); /* ... and return with status */
2006 : RETURN(MNRS_BEGNLEND);
2007 : END;
2008 :
```

```
2009 : /*
2010 : /*      Get information about the monitored system.
2011 : /*/
2012 :
2013 : IF M->MRBSV_PLAYBACK /* PLAYBACK request */
2014 : THEN DO;
2015 :     CALL READ_INPUT(NEXT_REC); /* Read system information record */
2016 :     IF MC->MCASV_EOF /* If end-of-file, */
2017 :     THEN DO;
2018 :         CALL MON_ERR(MNRS_PREMEOF); /* Can't find sys info record; log the error */
2019 :         RETURN (MNRS_PREMEOF); /* ... and return to caller */
2020 :     END;
2021 :
2022 :     TEMP_PTR = MC->MCASA_INPUT_PTR; /* Establish ptr to sys info record */
2023 :     TEMP_TYPE = UNSPEC(SYI_TYPE); /* Get sys info type into a byte for compare */
2024 :     IF TEMP_PTR->MNR_SYISB_TYPE ^= TEMP_TYPE /* If this record is not the sys info rec, */
2025 :     THEN DO;
2026 :         CALL MON_ERR(MNRS_INVINPFIL); /* Log an error */
2027 :         RETURN(MNRS_INVINPFIL); /* ... and return to caller */
2028 :     END;
2029 :
2030 : /*
2031 : /* Move entire sys info record to System Information Area
2032 : /*/
2033 :
2034 : SPTR->MNR_SYISB_TYPE = TEMP_PTR->MNR_SYISB_TYPE; /* Get SYI type code */
2035 : SPTR->MNR_SYISW_FLAGS = TEMP_PTR->MNR_SYISW_FLAGS; /* Get all flags */
2036 : SPTR->MNR_SYISB_MPCPUS = TEMP_PTR->MNR_SYISB_MPCPUS; /* Get number of cpu's */
2037 : SPTR->MNR_SYISW_MAXPRCCT = TEMP_PTR->MNR_SYISW_MAXPRCCT; /* Get MAXPROCESSCNT SYSGEN parameter */
2038 : SPTR->MNR_SYISQ_BOOTTIME = TEMP_PTR->MNR_SYISQ_BOOTTIME; /* Get system boot time */
2039 :
2040 : IF MNR_SYISK_NODENAME < MC->MCASL_INPUT_LEN /* If NODENAME field is defined for input file, */
2041 : THEN SPTR->MNR_SYIST_NODENAME = TEMP_PTR->MNR_SYIST_NODENAME; /* ... then pick it up from there */
2042 : ELSE UNSPEC(SPTR->MNR_SYIST_NODENAME) = '0'B; /* Otherwise, simply clear it */
2043 :
2044 : IF MNR_SYISK_BALSETMEM < MC->MCASL_INPUT_LEN /* If BALSETMEM field is defined for input file, */
2045 : THEN SPTR->MNR_SYISL_BALSETMEM = TEMP_PTR->MNR_SYISL_BALSETMEM; /* ... then pick it up from there */
2046 : ELSE SPTR->MNR_SYISL_BALSETMEM = BALSETMEM_DEF; /* Otherwise, use a constant default value */
2047 :
2048 : IF MNR_SYISK_MPWHILIM < MC->MCASL_INPUT_LEN /* If MPWHILIM field is defined for input file, */
2049 : THEN SPTR->MNR_SYISL_MPWHILIM = TEMP_PTR->MNR_SYISL_MPWHILIM; /* ... then pick it up from there */
2050 : ELSE SPTR->MNR_SYISL_MPWHILIM = MPWHILIM_DEF; /* Otherwise, use a constant default value */
2051 :
2052 : IF MNR_SYISK_CPUTYPE < MC->MCASL_INPUT_LEN /* If CPUTYPE field is defined for input file, */
2053 : THEN SPTR->MNR_SYISL_CPUTYPE = TEMP_PTR->MNR_SYISL_CPUTYPE; /* ... then pick it up from there */
2054 : ELSE SPTR->MNR_SYISL_CPUTYPE = 0; /* Otherwise, simply clear */
2055 :
2056 : END;
2057 :
```



```
2058  
2059 :  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085
```

```
ELSE DO;  
    SPTR->MNR_SYISB_TYPE = UNSPEC(SYI_TYPE);  
    SPTR->MNR_SYISV_RESERVED1 = '0'B;  
    SPTR->MNR_SYISV_FILLER = '0'B;  
    IF MPCHECK()  
    THEN SPTR->MNR_SYISB_MPCPUS = 2;  
    ELSE SPTR->MNR_SYISB_MPCPUS = 1;  
    SPTR->MNR_SYISW_MAXPRCCT = SGN$GW_MAXPRCCT;  
    CALL = COMPUTE_BOOTTIME();  
    IF STATUS = NOT_SUCCESSFUL  
    THEN DO;  
        CALL MON_ERR(MNR$_UNEXPERR,CALL);  
        RETURN(MNR$_UNEXPERR);  
    END;  
  
    CALL = CLUS_NET_INFO();  
    IF STATUS = NOT_SUCCESSFUL  
    THEN DO;  
        CALL MON_ERR(MNR$_UNEXPERR,CALL);  
        RETURN(MNR$_UNEXPERR);  
    END;  
  
    SPTR->MNR_SYISL_BALSETMEM = PFN$GL_PHYPGCNT;  
    SPTR->MNR_SYISL_MPWHILIM = MPW$GW_HILIM;  
  
END;
```

```
/* LIVE request */  
/* Fill the System Information Area from the running system */  
/* Get SYI type code */  
/* Clear reserved flag ... */  
/* ... and all unused flags */  
/* Multiprocessing capability? */  
/* Yes -- 2 cpu's */  
/* No -- just 1 cpu */  
/* Get MAXPROCESSCNT SYSGEN parameter */  
/* Get system time at boot into MNR_SYISQ_BOOTTIME */  
/* Failed? */  
  
/* Yes -- log the error */  
/* ... and return with status */  
  
/* Get cluster and network info (incl CPU type) into SYI */  
/* Failed? */  
  
/* Yes -- log the error */  
/* ... and return with status */  
  
/* Get balance set memory size (in pages) */  
/* Get MPW_HILIMIT SYSGEN parameter */
```

```
2086 | 2086 | /*  
2087 | 2087 | /* If PLAYBACK, read first class record from input file  
2088 | 2088 | /* to "prime the pump."  
2089 | 2089 | /*/  
2090 |  
2091 | IF M->MRBSV_PLAYBACK  
2092 | THEN DO;  
2093 |     CALL READ_INPUT(SKIP_TO_CLASS); /* Set up first class record for COLLECTION_EVENT rtn */  
2094 |     IF MC->MCASV_EOF /* If end-of-file, */  
2095 |     THEN DO;  
2096 |         CALL MON_ERR(MNRS_PREMEOF); /* Log the error */  
2097 |         RETURN (MNRS_PREMEOF); /* ... and return to caller */  
2098 |     END;  
2099 |  
2100 | /*  
2101 | /* If a future playback request, read input file, skipping  
2102 | /* past class records until file is positioned to requested  
2103 | /* begin point. Examine file time value only for the first  
2104 | /* class record within an interval, to ensure that the request  
2105 | /* will begin at an interval boundary. If end-of-file is hit  
2106 | /* during this operation, terminate the request with an error.  
2107 | /*/  
2108 |  
2109 | IF MC->MCASV_FUTURE  
2110 | THEN DO;  
2111 |     F = MC->MCASV_INPUT_PTR;  
2112 |     DO WHILE (^ MC->MCASV_EOF & QUAD_LT_QUAD(F->MNR_CLSSQ_STAMP,M->MRBSQ_BEGINNING));  
2113 |  
2114 |     READ FILE(INPUT_FILE) INTO(INPUT_DATA); /* Read rec following first class record */  
2115 |  
2116 |     DO WHILE (^ MC->MCASV_EOF & F->MNR_CLSSB_TYPE ^= MC->MCASB_FIRSTC);  
2117 |  
2118 |     READ FILE(INPUT_FILE) INTO(INPUT_DATA); /* Read until first class found again */  
2119 |  
2120 |     END;  
2121 |     END;  
2122 |     IF MC->MCASV_EOF /* EOF => bad beginning time */  
2123 |     THEN DO;  
2124 |         CALL MON_ERR(MNRS_BEGAN); /* Log the error */  
2125 |         RETURN(MNRS_BEGAN); /* ... and return with status */  
2126 |     END;  
2127 |     END;  
2128 |  
2129 |     MC->MCASL_INPUT_LEN = LENGTH(INPUT_DATA); /* Establish length of input */  
2130 |     END;  
2131 |  
2132 | RETURN(NORMAL); /* Return to caller */  
2133 | END REQUEST_INIT;  
2134 |
```



```
2135 1 RECORD_INIT: Procedure Returns(fixed binary(31));
2136 2
2137 2 /*
2138 2 /*++
2139 2 /*
2140 2 /* FUNCTIONAL DESCRIPTION:
2141 2 /*
2142 2 /* RECORD_INIT
2143 2 /*
2144 2 /* Called by EXECUTE_REQUEST to open the output (recording) file.
2145 2 /*
2146 2 /* INPUTS:
2147 2 /*
2148 2 /* None
2149 2 /*
2150 2 /* OUTPUTS:
2151 2 /*
2152 2 /* None
2153 2 /*
2154 2 /* ROUTINE VALUE:
2155 2 /*
2156 2 /* SSS_NORMAL
2157 2 /*
2158 2 /*--
2159 2 /*/
2160 2
2161 2 /*
2162 2 /*
2163 2 /*
2164 2 /* LOCAL STORAGE
2165 2 /*
2166 2 /*
2167 2 /*/
2168 2
2169 2 %INCLUDE PLI_FILE_DISPLAY;
2308 2
2309 2 Declare
2310 2 RECORD_EXPTR POINTER,
2311 2 TEMP_PTR POINTER,
2312 2 01 TEMP_BASED(TEMP_PTR),
2313 2 02 L FIXED_BINARY(15),
2314 2 02 DC FIXED_BINARY(15),
2315 2 02 A POINTER,
2316 2 TEMP_STR CHAR(TEMP.L) BASED(TEMP.A);
2317 2
2318 2 RECCT = 0; /* Init count of records written */
2319 2 M->MRBSV REC CL REQ = YES; /* Indicate record cleanup is required */
2320 2 CLOSE FILE(RECORD_FILE); /* Make sure file is closed before opening */
2321 2 TEMP_PTR = M->MRBSA RECORD; /* Set up ptr to output file name string */
2322 2 OPEN FILE(RECORD_FILE) OUTPUT TITLE(TEMP_STR) /* Open the output recording file */
2323 2 ENVIRONMENT(MAXIMUM_RECORD_SIZE(MAX_REC_SIZE), /* such that others may read it */
2324 2 SHARED READ);
2325 2 ALLOCATE PLI_FILE_DISPLAY SET (RECORD_EXPTR); /* Allocate space for the DISPLAY output */
2326 2 CALL DISPLAY(RECORD_FILE,RECORD_EXPTR->PLI_FILE_DISPLAY); /* Get the expanded file name */
2327 2 RECORD_STR = RECORD_EXPTR->EXPANDED_TITLE; /* Move the expanded string into global area for the module */
2328 2 FREE RECORD_EXPTR->PLI_FILE_DISPLAY; /* Release the storage area since the expanded string has be
```

EXECUTE_REQUEST
V04-000

J 8
16-SEP-1984 02:15:30
5-SEP-1984 15:10:53

VAX-11 PL/I X2.1-273 Page 32
ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PL1;1 (26)

```

2329      2
2330      2      RETURN(NORMAL);
2331      2      END RECORD_INIT;
2332      1

```

EXI VOI



```
2333 1  MONITOR_REQUEST: Procedure Returns(Fixed Binary(31));
2334 2
2335 3  /*
2336 4  /*      Execute first collection event. If live, collection events will
2337 5  /*      continue at AST level.
2338 6  /*/
2339 7
2340 8  IF ^ (MC->MCASB_FIRSTC = PROCS_CLSNO & M->MRBSV_PLAYBACK) /* If not playback of PROCESSES */
2341 9      THEN DO;
2342 10      CALL = SYSSDCLAST(COLLECTION_EVENT,,); /* ... then execute first collection event */
2343 11      IF STATUS = NOT_SUCCESSFUL /* $DCLAST failure? */
2344 12      THEN DO;
2345 13      CALL MON_ERR(MNRS_SSERROR,CALL,DCLAST_STR); /* Yes -- log the error */
2346 14      RETURN(MNRS_SSERROR); /* ... and return with status */
2347 15      END;
2348 16      END;
2349 17
2350 18  /*
2351 19  /*      Main monitoring loop. For playback, alternate collection and display events.
2352 20  /*      For live, simply issue display events in a loop while collection events loop
2353 21  /*      at AST level.
2354 22  /*/
2355 23
2356 24  DO WHILE (COLLENDED = NO); /* Loop while collection has not ended */
2357 25  IF M->MRBSV_PLAYBACK /* If this is a PLAYBACK request, */
2358 26      THEN DO;
2359 27      CALL = SYSSDCLAST(COLLECTION_EVENT,,); /* ... then execute a collection event */
2360 28      IF STATUS = NOT_SUCCESSFUL /* $DCLAST failure? */
2361 29      THEN DO;
2362 30      CALL MON_ERR(MNRS_SSERROR,CALL,DCLAST_STR); /* Yes -- log the error */
2363 31      RETURN(MNRS_SSERROR); /* ... and return with status */
2364 32      END;
2365 33      IF MC->MCASV_MULTFND & M->MRBSV_DISPLAY /* If multiple found and display requested, */
2366 34      & COLL_STATUS = NORMAL /* ... and collection_event finished OK, */
2367 35      THEN DO;
2368 36      CALL = DISPLAY_EVENT(); /* Execute a display event */
2369 37      IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Return if bad status */
2370 38
2371 39      IF COLLENDED = NO & M->MRBSV_DISP_TO_FILE = NO /* If still collecting, and displaying to SYSS$OUTPU
2372 40      THEN CALL = SYSSWFLOP(0,DISP_EV_FLAG_M ; REFR_EV_FLAG_M);
2373 41      /* ... then wait for viewing time or refresh request */
2374 42
2375 43      END;
2376 44      END;
2377 45
2378 46  ELSE DO; /* This is a LIVE request */
2379 47      IF M->MRBSV_DISPLAY /* If display requested */
2380 48      THEN DO;
2381 49      CALL = DISPLAY_EVENT(); /* ... then execute a display request */
2382 50      IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Return if bad status */
2383 51      END;
2384 52
2385 53      IF COLLENDED = NO /* Wait -- If no display, will wait whole request, */
2386 54      THEN CALL = SYSSWFLOP(0,DISP_EV_FLAG_M ; REFR_EV_FLAG_M);
2387 55
2388 56      END; /* ... while collection continues at AST level */
```

EXECUTE REQUEST
V04-000

2389 3 END;
2390 2

16-SEP-1984 02:15:31
5-SEP-1984 15:10:53

VAX-11 PL/I X2.1-273 Page 34
ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PL1;1 (27)



EXECUTE_REQUEST
V04-000

M 8
16-SEP-1984 02:15:31
5-SEP-1984 15:10:53

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PLI;1 (28)

Page 35

```
2391      2  
2392      2  
2393      2  
2394      2  
2395      2  
2396      2  
2397      2  
2398      2  
2399      1  
  
/*  
/*      End of main monitoring loop  
/*/  
  
RETURN(COLL_STATUS);  
  
END MONITOR_REQUEST;
```

/* Return with status from COLLECTION_EVENT */

EX
VO

```
2400 1 REQUEST_SUMMARY: Procedure Returns(Fixed Binary(31));
2401 2
2402 3 /*
2403 4 /* Since the MONITOR request has terminated (except for SUMMARY),
2404 5 /* certain CLEANUP routines may be executed now. Since SUMMARY
2405 6 /* output uses the same SYSS$OUTPUT stream through the SCRPKG as
2406 7 /* DISPLAY output, DISPLAY_CLEANUP MUST be done now.
2407 8 /*/
2408
2409 IF M->MRBSV_RECORD & M->MRBSV_REC_CL_REQ /* If this is a RECORD request AND cleanup required, */
2410 THEN CALL = RECORD_CLEANUP(); /* ... then do record cleanup */
2411 IF M->MRBSV_PLAYBACK & M->MRBSV_INP_CL_REQ /* If this is a PLAYBACK request AND cleanup required, */
2412 THEN CALL = INPUT_CLEANUP(); /* ... then do cleanup for it */
2413 IF M->MRBSV_DISPLAY & M->MRBSV_DIS_CL_REQ /* If this is a DISPLAY request AND cleanup required, */
2414 THEN CALL = DISPLAY_CLEANUP(); /* ... then do display cleanup */
2415
2416 CALL = SUMMARY_INIT(); /* Perform summary init */
2417 IF STATUS = NOT_SUCCESSFUL /* Failed? */
2418 THEN DO;
2419 CALL MON_ERR(MNR$ DISPERR,CALL); /* Yes -- log the error */
2420 RETURN(MNR$ DISPERR); /* ... and return with status */
2421 END;
2422 CALL = SUMMARY_EVENT(); /* Perform summarization */
2423 IF STATUS = NOT_SUCCESSFUL /* If failed, then return with status */
2424 THEN RETURN(CALL);
2425
2426 RETURN(NORMAL); /* Return to caller */
2427
2428 END REQUEST_SUMMARY;
2429
```



```
2430 1 DISPLAY_EVENT: Procedure Returns(fixed binary(31));
2431 2
2432 2 /*
2433 2 /*++
2434 2 /*
2435 2 /* FUNCTIONAL DESCRIPTION:
2436 2 /*
2437 2 /* DISPLAY_EVENT
2438 2 /*
2439 2 /* Called by EXECUTE_REQUEST to perform a single display event.
2440 2 /* One display event consists of creating and writing a screen
2441 2 /* image, including template if necessary, for a single class.
2442 2 /* The current class to be displayed is indicated within the
2443 2 /* DISPLAY_EVENT routine by the CURR DCLASS variable. CURR DCLASS
2444 2 /* is updated on each entry to DISPLAY_EVENT to indicate the
2445 2 /* next class in the list of requested classes. This causes the
2446 2 /* displays to cycle. DISPLAY_EVENT is entered once per viewing
2447 2 /* interval, or whenever a CTRL-W (screen refresh) is received.
2448 2 /*
2449 2 /* INPUTS:
2450 2 /*
2451 2 /* None
2452 2 /*
2453 2 /* OUTPUTS:
2454 2 /*
2455 2 /* None
2456 2 /*
2457 2 /* ROUTINE VALUE:
2458 2 /*
2459 2 /* SS$_NORMAL, or failing MONITOR status code.
2460 2 /*
2461 2 /*--
2462 2 /*/
2463 2
```

```

2464  /*
2465  /*
2466  /*
2467  /*
2468  /*
2469  /*
2470  /*
2471  /*
2472  Declare
2473  ADV_HOM_ITEM  ENTRY (POINTER);
2474
2475  Declare
2476  EV_FLAGS      BIT(32) ALIGNED,
2477  $$$_WASCLR    FIXED BINARY(31) GLOBALREF VALUE;
2478
2479  Declare
2480  DCDB          POINTER STATIC,
2481  COLL_TIME     BIT(64) ALIGNED STATIC,
2482  TEMP          FIXED BINARY(15);
2483
2484  Declare
2485  1 TIME_PARMS  STATIC,
2486  2 DATE_LEN    FIXED BINARY(31) INIT(11),
2487  2 DATE_PTR    POINTER,
2488  2 TIME_LEN    FIXED BINARY(31) INIT(8),
2489  2 TIME_PTR    POINTER,
2490
2491  DATE_OUT      CHAR(11) STATIC,
2492  TIME_OUT      CHAR(8) STATIC,
2493
2494  1 TIME_STR    GLOBALREF,
2495  2 L           FIXED BINARY(7),
2496  2 S           CHAR(1),
2497
2498  1 SYS_TIME_STR GLOBALREF,
2499  2 L           FIXED BINARY(7),
2500  2 S           CHAR(1);
2501
2502  Declare
2503  DATA_STR     CHAR(1) BASED(DCDB->CDB$A_FAOCTR),
2504  FAOSTK        FIXED BINARY(31) GLOBALREF;
2505
2506  Declare
2507  VIDEO_IND     BIT(1) ALIGNED;
2508

```



```
2509 2 CALL = SYSS$REDEF(DISP EV FLAG, EV_FLAGS); /* Examine state of display event flag */
2510 2 IF STATUS = NOT_SUCCESSFUL /* Failed? */
2511 2 THEN DO;
2512 2 CALL MON_ERR(MNRS$SSERROR, CALL, REDEF_STR); /* Yes -- log the error */
2513 2 RETURN(MNRS$SSERROR); /* ... and return with status */
2514 2 END;
2515 2
2516 2 IF CALL = SS$WASCLR /* If display event flag was clear, */
2517 2 THEN DO; /* (Assume this is a refresh event) */
2518 2 CALL = SYSS$CLREF(REFR EV_FLAG); /* Clear refresh event flag */
2519 2 MC->MCASV_REFRESH = YES; /* ... and indicate this is a refresh display event */
2520 2 IF STATUS = NOT_SUCCESSFUL /* SYSS$CLREF service call failed? */
2521 2 THEN DO;
2522 2 CALL MON_ERR(MNRS$SSERROR, CALL, CLREF_STR); /* Yes -- log the error */
2523 2 RETURN(MNRS$SSERROR); /* ... and return with status */
2524 2 END;
2525 2 END;
2526 2
2527 2 IF CURR_DCLASS = 0 & (DISPLAYING = NO ; MC->MCASV_REFRESH = YES) /* If class data not yet displayed, AND */
2528 2 /* ... first time thru or refresh requested */
2529 2 THEN DO;
2530 2 VIDEO_IND = MC->MCASV_VIDEO; /*...Get Video indicator */
2531 2 CALL = DISP_TEMPLATE(D_CURR_CDBPTR(1), VIDEO_IND); /* ... display a template for the first class, */
2532 2 /* ... forcing output to screen if video terminal */
2533 2 DISPLAYING = YES; /* Indicate that display output has begun */
2534 2 IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Check call */
2535 2 END;
2536 2
2537 2 IF CURR_DCLASS = MC->MCASW_DCLASSCT /* If did final class on previous entry, */
2538 2 THEN TEMP = 1; /* ... then start over at first one */
2539 2 ELSE TEMP = CURR_DCLASS + 1; /* ... otherwise, advance to next class */
2540 2 IF MC->MCASL_COLLCNT >= 2 ; D_CURR_CLASS_NO(TEMP) = PROCS_CLSNO /* If at least 2 collections have passed OR ... */
2541 2 /* ... this is the PROCESSES class, */
2542 2 THEN DO;
2543 2 IF ^ REPT_TOP /* If not the special TOP repeat, */
2544 2 THEN IF ADVANCE_DCLASS() = YES /* Test if display class should be advanced */
2545 2 THEN CURR_DCLASS = TEMP; /* ... and advance it accordingly */
2546 2
2547 2 DCDB = D_CURR_CDBPTR(CURR_DCLASS); /* Get CDB for current display class */
2548 2
2549 2 IF MC->MCASL_DISPCNT ^= 0 & (MC->MCASV_REFRESH = YES ; MC->MCASW_DCLASSCT ^= 1) & ^ REPT_TOP /* If template not printed above, AND ... */
2550 2 /* ... refresh requested OR more than 1 class, */
2551 2 /* ... AND not the special TOP repeat */
2552 2 THEN DO;
2553 2 CALL = DISP_TEMPLATE(DCDB, NO); /* Display template */
2554 2 IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Check call */
2555 2 END;
2556 2
2557 2 REPT_TOP = NO; /* Eliminate future TOP repeat */
2558 2 IF MC->MCASL_DISPCNT = 0 & D_CURR_CLASS_NO(CURR_DCLASS) = PROCS_CLSNO & DCDB->CDB$B_ST ^= REG_PROC /* If 1st TOP display, allow a 2nd consec TOP */
2559 2 THEN REPT_TOP = YES;
2560 2
2561 2 IF CTRLCZ_HIT = NO ; M->MRBSV_DISP_TO_FILE THEN /* If CTRL-C and Z not hit OR displaying to a file, */
2562 2 DO; /* ... then prepare to display actual data */
2563 2 IF DCDB->CDB$V_HOMOG THEN CALL ADV_HOM_ITEM(DCDB); /* If homog class, advance to next display item */
2564 2
```

EXECUTE_REQUEST
V04-000

E 9
16-SEP-1984 02:15:33
5-SEP-1984 15:10:53

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PLI;1 (32)

Page 40

```
2565      4      CALL = SYSS$SETAST(DISABLE_AST);          /* Disable collection events while filling display b
2566      4      CALL = FILL_DISP_BUFF(DCDB,COLL_TIME);      /* Fill display buffer for this class */
2567      4      CALL = SYSS$SETAST(ENABLE_AST);              /* Re-enable collection events */
2568      4
2569      4      /*
2570      4      /*
2571      4      /* Call DISPLAY_PUT to first display the date and time of the most recent collection,
2572      4      /* then to display the actual data itself.
2573      4      /*
```



```
2574 4 CALL = SYSSASCTIM(,DATE_OUT,COLL_TIME,0); /* Get ASCII date */
2575 4 CALL = SYSSASCTIM(,TIME_OUT,COLL_TIME,1); /* Get ASCII time */
2576 4 DATE_PTR = ADDR(DATE_OUT); /* Address of date string into FAOL list */
2577 4 TIME_PTR = ADDR(TIME_OUT); /* Address of time string into FAOL list */
2578 4 FAOL_REQUESTED = YES; /* Run it through FAOL */
2579 4 OUTP_REQUESTED = NO; /* ... but don't output it yet */
2580 4 IF DCDB->CDB$V_SYSCLS & DCDB->CDB$B_ST ^= ALL_STAT /* If special SYSTEM screen, issue it one way, */
2581 4 THEN DO; /* Length of time control string */
2582 4 PUT_LEN = SYS_TIME_STR.L; /* Length of time control string */
2583 4 CALC = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SYS_TIME_STR.S,TIME_PARMS); /* Send date and time to SCRPKG */
2584 4 ;
2585 4 END;
2586 4 ELSE DO; /* Otherwise issue it another way */
2587 4 PUT_LEN = TIME_STR.L; /* Length of time control string */
2588 4 CALC = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,TIME_STR.S,TIME_PARMS); /* Send date and time to SCRPKG */
2589 4 ;
2590 4 END;
2591 4 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2592 4 ;
2593 4 /*
2594 4 /* Put actual display data
2595 4 /*
2596 4 ;
2597 4 IF DCDB->CDB$V_STD /* Is this a standard class? */
2598 4 THEN /* Standard Class */
2599 4 IF DCDB->CDB$V_HOMOG /* Check type of standard class */
2600 4 THEN DO; /* Homogeneous Standard Class */
2601 4 CALL = DISPLAY_HOMOG(DCDB); /* Send homog data display lines to SCRPKG */
2602 4 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2603 4 END;
2604 4 ELSE DO; /* Heterogeneous Standard Class */
2605 4 FAOL_REQUESTED = YES; /* Run it through FAOL */
2606 4 OUTP_REQUESTED = YES; /* Output it now */
2607 4 CALL = DISPLAY_PUT(DPUT_FLAGS,DCDB->CDB$L_FAOCTR,DATA_STR,FAOSTK); /* Send display data to SCRPKG */
2608 4 ; /* Check status */
2609 4 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2610 4 END;
2611 4 ELSE /* Non-standard Class (PROCESSES) */
2612 4 IF DCDB->CDB$B_ST = REG_PROC /* Regular PROCESSES display */
2613 4 THEN DO; /* Regular PROCESSES display */
2614 4 CALL = DISPLAY_PROCS(DCDB,COLL_TIME); /* Send process display lines to SCRPKG */
2615 4 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2616 4 END;
2617 4 ELSE DO; /* TOP PROCESSES display */
2618 4 CALL = DISPLAY_TOP(DCDB); /* Send top process display lines to SCRPKG */
2619 4 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2620 4 END;
2621 4 MC->MCASL_DISPCNT = MC->MCASL_DISPCNT + 1; /* Count this display event */
2622 4 END;
2623 4 END;
2624 4 ;
2625 4 IF MC->MCASV_REFRESH THEN CALL = SYSSCANTIM(DISP_EV_FLAG,); /* If a refresh event, cancel "regular" display time
2626 4 ;
2627 4 IF COLLENDED = NO & ^ (M->MRBSV_PLAYBACK & M->MRBSV_DISP_TO_FILE) /* If collection still going, ... */
2628 4 ; /* ... AND not playing back to a file, */
2629 4 ;
```

EXECUTE_REQUEST
V04-000

6 9
16-SEP-1984 02:15:34
5-SEP-1984 15:10:53

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PLI;1 (33) Page 42

```
2630      2      THEN DO;  
2631      3      CALL = SYS$SETIMR(DISP_EV_FLAG,VIEWING_DEL,,DISP_EV_FLAG); /* Set flag when ready to display again */  
2632      3      IF STATUS = NOT_SUCCESSFUL /* Failed? */  
2633      3      THEN DO;  
2634      4      CALL MON_ERR(MNRS_SSERROR,CALL,SETIMR_STR); /* Yes -- log the error */  
2635      4      RETURN(MNRS_SSERROR); /* ... and return with status */  
2636      4      END;  
2637      3      END;  
2638      2  
2639      2      MC->MCASV_REFRESH = NO; /* Indicate not a refresh display event for next tim  
2640      2  
2641      2      RETURN(NORMAL);  
2642      2  
2643      2
```



```
2644 ADVANCE_DCLASS: Procedure Returns(Bit(1) aligned);          /* Test if display class should be advanced */
2645
2646 /*
2647 /*++
2648 /*
2649 /* FUNCTIONAL DESCRIPTION:
2650 /*
2651 /*     ADVANCE_DCLASS
2652 /*
2653 /*     This routine checks whether the current display class
2654 /*     (as indicated in the variable CURR_DCLASS) should be
2655 /*     advanced to the next requested class, or left where
2656 /*     it is. Normally, the class is advanced, but in the case
2657 /*     where the current class is homogeneous and not yet at
2658 /*     the end of its item list, the class is not advanced.
2659 /*
2660 /* INPUTS:
2661 /*
2662 /*     None
2663 /*
2664 /* OUTPUTS:
2665 /*
2666 /*     None
2667 /*
2668 /* ROUTINE VALUE:
2669 /*
2670 /*     YES if the current display class should be advanced.
2671 /*     NO  otherwise
2672 /*
2673 /*--
2674 /*/
2675
2676 /*
2677 /*     ┌──────────────────────────────────────────────────────────┐
2678 /*     │                                                                │
2679 /*     │                                LOCAL STORAGE                                │
2680 /*     │                                                                │
2681 /*     └──────────────────────────────────────────────────────────┘
2682 /*/
2683
2684 Declare
2685     ADVANCE_CLASS BIT(1) ALIGNED,          /* YES => advance display class */
2686     RCDB          POINTER;                 /* CDB pointer for most recent class */
2687
2688 ADVANCE_CLASS = YES;                      /* Assume class will be advanced */
2689 IF CURR_DCLASS ^= 0                      /* If not the first display event, */
2690     THEN DO;
2691         RCDB = D CURR_CDBPTR(CURR_DCLASS); /* get CDB addr for most recent display event */
2692         IF RCDB->CDB$V HOMOG              /* check if it is a homogeneous class */
2693             THEN IF RCDB->CDB$A CDX->CDX$B IDISCONSEC < RCDB->CDB$A_CDX->CDX$B_IDISCONSEC /* All items displayed? */
2694                 THEN ADVANCE_CLASS = NO; /* No -- don't advance */
2695     END;
2696
2697 RETURN(ADVANCE_CLASS);                  /* Return with indicator */
2698
2699 END ADVANCE_DCLASS;
```

EXECUTE REQUEST
V04-000

16-SEP-1984 02:15:35
5-SEP-1984 15:10:53

VAX-11 PL/I X2.1-273 Page 44
ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PL1;1 (34)

```
2700      2
2701      2      END DISPLAY_EVENT;
2702      1
```

EXE
V04

[illegible]


```
2703 1 SUMMARY_EVENT: Procedure Returns(fixed binary(31));
2704 2
2705 3 /*
2706 4 /*++
2707 5 /*
2708 6 /* FUNCTIONAL DESCRIPTION:
2709 7 /*
2710 8 /* SUMMARY_EVENT
2711 9 /*
2712 10 /* Called by EXECUTE_REQUEST once per request to create a
2713 11 /* summary file containing a screen image for each of the
2714 12 /* requested classes.
2715 13 /*
2716 14 /* INPUTS:
2717 15 /*
2718 16 /* None
2719 17 /*
2720 18 /* OUTPUTS:
2721 19 /*
2722 20 /* None
2723 21 /*
2724 22 /* ROUTINE VALUE:
2725 23 /*
2726 24 /* SSS_NORMAL, or failing MONITOR status code.
2727 25 /*
2728 26 /*--
2729 27 /*/
2730 28
```

```
2731 1 2 /*
2732 2 2 /*
2733 3 2 /*
2734 4 2 /*
2735 5 2 /*
2736 6 2 /*
2737 7 2 /*/
2738 8 2
2739 9 2 Declare
2740 10 2 SUMMARY_TOP ENTRY (POINTER) /* MACRO-32 rtn to set up for TOP summary */
2741 11 2 RETURNS (FIXED BINARY(31)),
2742 12 2 ADV_HOM_ITEM ENTRY (POINTER); /* MACRO-32 rtn to advance homog class to next displ
2743 13 2
2744 14 2 Declare
2745 15 2 DCDB POINTER STATIC, /* CDB for current display class */
2746 16 2 COLL_TIME BIT(64) ALIGNED STATIC; /* Time stamp from most recent collection */
2747 17 2
2748 18 2 Declare
2749 19 2 1 SUMM_PARMS STATIC, /* FAOL parms for summary beg and end date/times */
2750 20 2 2 BEG_LEN FIXED BINARY(31) INIT(20), /* Length of beginning date/time string */
2751 21 2 2 BEG_PTR POINTER, /* Pointer to beginning date/time string */
2752 22 2 2 END_LEN FIXED BINARY(31) INIT(20), /* Length of ending date/time string */
2753 23 2 2 END_PTR POINTER, /* Pointer to ending date/time string */
2754 24 2
2755 25 2 BEG_OUT CHAR(23) STATIC, /* Beg date/time output string from ASCTIM */
2756 26 2 END_OUT CHAR(23) STATIC, /* End date/time output string from ASCTIM */
2757 27 2
2758 28 2 1 SUMMLINE_STR GLOBALREF, /* Summary date/time FAO control string */
2759 29 2 2 L FIXED BINARY(7), /* Length */
2760 30 2 2 S CHAR(1), /* First character of string */
2761 31 2
2762 32 2 1 SYS_SUMMLINE_STR GLOBALREF, /* Summary date/time FAO control string for SYSTEM c
2763 33 2 2 L FIXED BINARY(7), /* Length */
2764 34 2 2 S CHAR(1); /* First character of string */
2765 35 2
2766 36 2 Declare
2767 37 2 DATA_STR CHAR(1) BASED(DCDB->CDB$A FAOCTR), /* First char of FAO ctr str for display data */
2768 38 2 FAOSTK FIXED BINARY(31) GLOBALREF; /* First longword of FAOL parm list */
2769 39 2
```

LOCAL STORAGE


```
2770      DO CURR_DCLASS = 1 TO MC->MCASW_DCLASSCT          /* Loop once for each requested class */
2771      WHILE (MC->MCASL_COLLCNT >= 2);                      /* ... but only if at least 2 collections */
2772
2773          DCDB = D_CURR_CDBPTR(CURR_DCLASS);               /* Get CDB for current display class */
2774          CALL = DISP_TEMPLATE(DCDB,NO);                  /* Send template to SCRPKG, but don't output yet */
2775          IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);  /* Check call */
2776          IF (D_CURR_CLASS_NO(CURR_DCLASS) = PROCS_CLSNO /* If PROCESSES class with TOP screen, */
2777              & DCDB->CDB$B_ST ^= REG_PROC)
2778              ! (DCDB->CDB$V_SYSCLS & DCDB->CDB$B_ST ^= ALL_STAT) /* ... OR SYSTEM class with single stat, */
2779              THEN CALL = SUMMARY_TOP(DCDB);              /* ... then do TOP setup */
2780
2781          IF DCDB->CDB$V_HOMOG                              /* If homogeneous class, */
2782          THEN DO;
2783              DCDB->CDB$A_CDX->CDX$B_IDISCONSEC = 0;        /* Init consec display item number */
2784              DO WHILE(DCDB->CDB$A_CDX->CDX$B_IDISCONSEC < DCDB->CDB$A_CDX->CDX$B_IDISCT);
2785
2786                  CALL ADV_HOM_ITEM(DCDB);                /* Advance to next display item */
2787                  CALL = SUMM_ONE_CLASS();                /* Summarize once for each item */
2788                  IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Check call */
2789              END;
2790          ELSE DO;
2791              CALL = SUMM_ONE_CLASS();                    /* Heterogeneous class or PROCESSES */
2792              IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Only need to call once */
2793          END;
2794      END;
2795      RETURN(NORMAL); /* Return */
2800
2801
```

```
2802 SUMM_ONE_CLASS: Procedure Returns(fixed binary(31));
2803
2804 /*
2805 /**+
2806 /*
2807 /* FUNCTIONAL DESCRIPTION:
2808 /*
2809 /*     SUMM_ONE_CLASS
2810 /*
2811 /*     Called by SUMMARY_EVENT to put screen images to the
2812 /*     summary file for a single class. For heterogeneous
2813 /*     classes, a single screen image is required. For
2814 /*     other classes, multiple screen images may be
2815 /*     required.
2816 /*
2817 /* INPUTS:
2818 /*
2819 /*     None
2820 /*
2821 /* OUTPUTS:
2822 /*
2823 /*     None
2824 /*
2825 /* ROUTINE VALUE:
2826 /*
2827 /*     SS$_NORMAL, or failing MONITOR status code.
2828 /*
2829 /*--
2830 /*/
2831
```



```
2832 CALL = FILL_DISP_BUFF(DCDB,COLL_TIME);          /* Fill display buffer for this class */
2833
2834
2835 /*
2836 /* Call DISPLAY_PUT to first display the summary time range,
2837 /* then to display the actual data itself.
2838 /*
2839
2840 CALL = SYSSASCTIM(BEG_OUT,M->MRBSQ BEGINNING,0); /* Get ASCII beginning time */
2841 CALL = SYSSASCTIM(END_OUT,COLL_TIME,0);          /* Get ASCII ending time */
2842 BEG_PTR = ADDR(BEG_OUT);                          /* Address of beg string into FAOL list */
2843 END_PTR = ADDR(END_OUT);                          /* Address of end string into FAOL list */
2844 FAOL_REQUESTED = YES;                             /* Run it through FAOL */
2845 OUTP_REQUESTED = NO;                             /* ... but don't output it yet */
2846 IF DCDB->CDB$V_SYSCLS & DCDB->CDB$B_ST ^= ALL_STAT /* If special SYSTEM screen, issue it a special way,
2847 THEN DO;
2848     PUT_LEN = SYS_SUMMLINE_STR.L;                  /* Length of SYSTEM summary control string */
2849     CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SYS_SUMMLINE_STR.S,SUMM_PARMS);
2850 /* Send summary line to SCRPKG */
2851 END;
2852 ELSE DO;
2853     PUT_LEN = SUMMLINE_STR.L;                      /* Length of summary control string */
2854     CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SUMMLINE_STR.S,SUMM_PARMS);
2855 /* Send summary line to SCRPKG */
2856 END;
2857 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);      /* Check status */
2858
2859 /*
2860 /* Put actual display data
2861 /*
2862
2863 IF DCDB->CDB$V_STD                                /* Is this a standard class? */
2864 THEN
2865     IF DCDB->CDB$V_HOMOG                          /* Standard Class */
2866     THEN DO;                                     /* Check type of standard class */
2867         CALL = DISPLAY_HOMOG(DCDB);              /* Homogeneous Standard Class */
2868         IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Send homog data display lines to SCRPKG */
2869         /* Check status */
2870     ELSE DO;                                     /* Heterogeneous Standard Class */
2871         FAOL_REQUESTED = YES;                    /* Run it through FAOL */
2872         OUTP_REQUESTED = YES;                    /* Output it now */
2873         CALL = DISPLAY_PUT(DPUT_FLAGS,DCDB->CDB$L_FAOCTR,DATA_STR,FAOSTK);
2874 /* Send display data to SCRPKG */
2875         IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2876     END;
2877 ELSE
2878     IF DCDB->CDB$B_ST = REG_PROC                  /* Non-standard Class (PROCESSES) */
2879     THEN DO;                                     /* Regular PROCESSES display */
2880         CALL = DISPLAY_PROCS(DCDB,COLL_TIME);    /* Send process display lines to SCRPKG */
2881         IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2882     ELSE DO;                                     /* TOP PROCESSES display */
2883         CALL = DISPLAY_TOP(DCDB);                /* Send top process display lines to SCRPKG */
2884         IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2885     END;
2886
2887
```



```

2888      3      RETURN(NORMAL);
2889      3
2890      3      END SUMM_ONE_CLASS;
2891      2
2892      2      END SUMMARY_EVENT;
2893      1

```

[illegible]


```

2894 1 SAVE_SUM_BUFFS: Procedure Returns(Fixed Binary(31)); /* Save SUM buffers into M.F. Summary Buffer */
2895 2 /* ... for all classes for the current input file */
2896 2
2897 2 /*
2898 2 /*++
2899 2 /*
2900 2 /* FUNCTIONAL DESCRIPTION:
2901 2 /*
2902 2 /*     SAVE_SUM_BUFFS
2903 2 /*
2904 2 /*     Called by EXECUTE REQUEST once per request to save the
2905 2 /*     SUM buffers of all classes into their respective Multi-File
2906 2 /*     Summary Blocks.
2907 2 /*
2908 2 /* IMPLICIT INPUTS:
2909 2 /*
2910 2 /*     MFSPTR -- Pointer to MFS (Multi-File Summary Block)
2911 2 /*
2912 2 /*     MFSSB_CUR_COL -- column number for column (on m.f. summary report) into which the
2913 2 /*                      data from the SUM buffers will be stored.
2914 2 /*
2915 2 /*     DISPLAY_CLASSES -- 128-bit string of classes to be summarized (excludes STATES, MODES
2916 2 /*                      and PROCESSES if they are present only in support of SYSTEM).
2917 2 /*
2918 2 /* OUTPUTS:
2919 2 /*
2920 2 /*     None
2921 2 /*
2922 2 /* ROUTINE VALUE:
2923 2 /*
2924 2 /*     SSS_NORMAL, or failing MONITOR status code.
2925 2 /*
2926 2 /*--
2927 2 /*/
2928 2

```

```
2929 | 2 /*
2930 | 2 /*
2931 | 2 /*
2932 | 2 /*
2933 | 2 /*
2934 | 2 /*
2935 | 2 /*
2936 | 2 /*
2937 | 2 Declare
2938 | 2 COL_NO          FIXED BINARY(7);          /* Column number to store sums into */
2939 | 2
2940 | 2 Declare
2941 | 2 ALLOC_SUMBUFS   ENTRY(BIT(128) ALIGNED)    /* MONITOR MACRO-32 rtn to allocate m.f. summary buf
2942 | 2 RETURNS (FIXED BINARY(31)),
2943 | 2 CAPTURE_SUMS     ENTRY (POINTER, FIXED BINARY(7)) /* MACRO-32 routine to move SUM buffer to M.F. Summa
2944 | 2 RETURNS (FIXED BINARY(31)),
2945 | 2 ADV_HOM_ITEM     ENTRY (POINTER);          /* MACRO-32 rtn to advance homog class to next displ
2946 | 2
2947 | 2 Declare
2948 | 2 DCDB            POINTER STATIC;            /* CDB for current class */
2949 | 2
2950 | 2 Declare
2951 | 2 DISPLAY_CLASSES BIT(128) ALIGNED GLOBALREF; /* Classes to be summarized */
2952 | 2
2953 | 2
```

LOCAL STORAGE


```
2954      2      CALL = ALLOC SUMBUFS(DISPLAY CLASSES);          /* Allocate m.f. summary buffers (if not done yet) */
2955      2      IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);    /* Check call */
2956      2
2957      2      COL_NO = MFSSB_CUR_COL;                             /* Get number of column currently being processed */
2958      2
2959      2      DO CURR_DCLASS = 1 TO MC->MCASW_DCLASSCT;          /* Loop once for each summarized class */
2960      2
2961      2          DCDB = D_CURR_CDBPTR(CURR_DCLASS);             /* Get CDB for current class */
2962      2
2963      2          IF DCDB->CDB$$_ECOUNT ^= 0                      /* If we have some elements, */
2964      2              THEN DO;                                     /*
2965      2                  CALL = CAPTURE SUMS(DCDB,COL_NO);        /* Capture SUM buffer for this class and "column" */
2966      2                  IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Check call */
2967      2                  END;
2968      2
2969      2      END;
2970      2
2971      2      RETURN(NORMAL);                                     /* Return */
2972      2
2973      2      END SAVE_SUM_BUFS;
2974      1
```

```

2975 1 /*
2976 1 /*++
2977 1 /*
2978 1 /* FUNCTIONAL DESCRIPTION:
2979 1 /*
2980 1 /*     CLEANUP Routines. RECORD_CLEANUP, SUMMARY_CLEANUP
2981 1 /*           INPUT_CLEANUP, and DISPLAY_CLEANUP
2982 1 /*
2983 1 /*     Called by EXECUTE_REQUEST to close files, reset terminal
2984 1 /*     characteristics, and release associated resources.
2985 1 /*     INPUT_CLEANUP can also be called by MFSUM_REQUEST to close
2986 1 /*     an input file and free the allocated buffer memory.
2987 1 /*     SUMMARY_CLEANUP can also be called by MFSUM_REQUEST to close
2988 1 /*     the summary file.
2989 1 /*
2990 1 /* INPUTS:
2991 1 /*
2992 1 /*     None
2993 1 /*
2994 1 /* OUTPUTS:
2995 1 /*
2996 1 /*     None
2997 1 /*
2998 1 /* ROUTINE VALUE:
2999 1 /*
3000 1 /*     SSS_NORMAL
3001 1 /*
3002 1 /*--
3003 1 /*/
3004 1
3005 1 REQUEST_CLEANUP: Procedure;
3006 2
3007 2 Declare
3008 2 FREE_MEM          ENTRY RETURNS(FIXED BINARY(31));          /* MONITOR MACRO-32 routine to issue LIB$FREE_VM's */
3009 2
3010 2 CALL = FREE_MEM();                                           /* Free virtual memory acquired for this request */
3011 2
3012 2 IF M->MRBSV_RECORD & M->MRBSV_REC_CL_REQ                     /* If this is a RECORD request AND cleanup required, */
3013 2 THEN CALL = RECORD_CLEANUP();                                /* ... then do record cleanup */
3014 2 IF M->MRBSV_PLAYBACK & M->MRBSV_INP_CL_REQ                   /* If this is a PLAYBACK request AND cleanup required, */
3015 2 THEN CALL = INPUT_CLEANUP();                                  /* ... then do cleanup for it */
3016 2 IF M->MRBSV_DISPLAY & M->MRBSV_DIS_CL_REQ                     /* If this is a DISPLAY request AND cleanup required, */
3017 2 THEN CALL = DISPLAY_CLEANUP();                                /* ... then do display cleanup */
3018 2 IF M->MRBSV_SUMMARY & M->MRBSV_SUM_CL_REQ                     /* If this is a SUMMARY request AND cleanup required, */
3019 2 THEN CALL = SUMMARY_CLEANUP();                                /* ... then do summary cleanup */
3020 2
3021 2 RETURN;
3022 2 END REQUEST_CLEANUP;
3023 1
3024 1
3025 1 RECORD_CLEANUP: Procedure Returns(fixed binary(31));
3026 1
3027 2 Declare
3028 2 H                POINTER;                                     /* Pointer to file header record */
3029 2
3030 2

```


G 10
16-SEP-1984 02:15:38 VAX-11 PL/I X2.1-273 Page 55
5-SEP-1984 15:10:53 ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PL1:1 (43)

```

3031 M->MRBSV REC CL REQ = NO; /* Indicate record cleanup is no longer required */
3032 CLOSE FILE(RECORD_FILE); /* Close the record file */
3033 IF RECCT > 0 /* If file is non-empty, */
3034 THEN DO: /* ... then want to re-write the header */
3035 OPEN FILE(RECORD_FILE) UPDATE TITLE(RECORD_STR) /* Re-open it to re-write header while */
3036 ENVIRONMENT(SHARED_READ); /* allowing others to read the file */
3037 READ FILE(RECORD_FILE) SET(H); /* Read header record */
3038 H->MNR_HDR$Q_ENDING = M->MRBSQ_ENDING; /* Update the ending time */
3039 H->MNR_HDR$Q_RECCT = RECCT; /* ... and the record count */
3040 REWRITE FILE(RECORD_FILE); /* Re-write the header record */
3041 CLOSE FILE(RECORD_FILE); /* ... and close it up again */
3042 END;
3043
3044 RETURN(NORMAL); /* Return */
3045 END RECORD_CLEANUP;
3046
3047 END EXECUTE_REQUEST;
3048

```

```
3049 SUMMARY_CLEANUP: Procedure Returns(fixed binary(31));
3050
3051 1 %INCLUDE MONDEF; /* Monitor utility structure definitions */
3819 1
3820 1 Declare
3821 1 LIB$SET_BUFFER ENTRY (ANY VALUE), /* Rtn to set and clear buffer mode for the SCRPKG */
3822 1 SCR$SET_CURSOR ENTRY (ANY VALUE, ANY VALUE), /* SCRPKG rtn to set the cursor position */
3823 1 SCR$UP_SCROLL ENTRY, /* SCRPKG rtn to scroll up one line */
3824 1 SCR$STOP_OUTPUT ENTRY; /* Rtn to stop SCRPKG output stream */
3825 1
3826 1 Declare
3827 1 NORMAL FIXED BINARY(31) GLOBALREF, /* MONITOR normal status value */
3828 1 MRBPTR POINTER GLOBALREF, /* Pointer to MRB (Monitor Request Block) */
3829 1 M POINTER DEFINED(MRBPTR), /* Synonym for MRBPTR */
3830 1
3831 1 1 BOT_CURS GLOBALREF, /* Place cursor on bottom of screen */
3832 1 2 L FIXED BINARY(7), /* Length */
3833 1 2 S CHAR(1), /* First character of string */
3834 1
3835 1 SFSPEC CHAR(8) BASED; /* Dummy summary file spec descriptor */
3836 1
3837 1 M->MRB$V_SUM_CL_REQ = NO; /* Indicate summary cleanup is no longer required */
3838 1 CALL LIB$SET_BUFFER(0); /* Indicate "clear buffer mode" to SCRPKG */
3839 1 /* ... and output what's left in the buffer */
3840 1 CALL SCR$SET_CURSOR(24,1); /* Place cursor on bottom line */
3841 1 CALL SCR$UP_SCROLL(); /* ... and scroll up one line */
3842 1 CALL SCR$STOP_OUTPUT(); /* Stop output stream and close summary file */
3843 1 RETURN(NORMAL); /* Return */
3844 1 END SUMMARY_CLEANUP;
3845
```



```
3846 INPUT_CLEANUP: Procedure Returns(fixed binary(31));
3847
3848 1
3849 1 %INCLUDE MONDEF; /* Monitor utility structure definitions */
4617 1
4618 1 Declare
4619 1 MAX_REC_SIZE FIXED BINARY(31) GLOBALREF VALUE, /* Max record size for PLAYBACK & RECORD files */
4620 1 NORMAL FIXED BINARY(31) GLOBALREF, /* MONITOR normal status value */
4621 1 MRBPTR POINTER GLOBALREF, /* Pointer to MRB (Monitor Request Block) */
4622 1 M POINTER DEFINED(MRBPTR), /* Synonym for MRBPTR */
4623 1 INPUT_CPTR POINTER GLOBALREF, /* Ptr to input buffer count word */
4624 1 INPUT_DATA CHAR(MAX_REC_SIZE) VARYING BASED(INPUT_CPTR); /* Playback file input buffer */
4625 1
4626 1 Declare
4627 1 INPUT_FILE FILE RECORD INPUT; /* Monitor Input (Playback) File */
4628 1
4629 1 M->MRBSV INP CL REQ = NO; /* Indicate input cleanup is no longer required */
4630 1 CLOSE FILE(INPUT_FILE); /* Close the input file */
4631 1 IF INPUT_CPTR ^= NULL() /* If input buffer had been acquired */
4632 1 THEN FREE INPUT_CPTR->INPUT_DATA; /* ... then free it */
4633 1 RETURN(NORMAL); /* Return */
4634 1 END INPUT_CLEANUP;
4635
4636 DISPLAY_CLEANUP: Procedure Returns(fixed binary(31));
4637
4638 1 %INCLUDE MONDEF; /* Monitor utility structure definitions */
5406 1
5407 1 Declare
5408 1 DISPLAYING BIT(1) ALIGNED GLOBALREF, /* YES=> display output is active */
5409 1 CTRLZ_HIT BIT(1) ALIGNED GLOBALREF, /* YES=> CTRL/Z has been hit */
5410 1 NORMAL FIXED BINARY(31) GLOBALREF, /* MONITOR normal return status */
5411 1 MRBPTR POINTER GLOBALREF, /* Pointer to MRB (Monitor Request Block) */
5412 1 M POINTER DEFINED(MRBPTR); /* Synonym for MRBPTR */
5413 1
5414 1 Declare
5415 1 LIB$SET_BUFFER ENTRY (ANY VALUE), /* Rtn to set and clear buffer mode for the SCRPKG */
5416 1 PUT_TO_SCREEN ENTRY (ANY VALUE, ANY), /* Rtn to put an arbitrary buffer to the SCRPKG */
5417 1 SCR$SET_CURSOR ENTRY (ANY VALUE, ANY VALUE), /* SCRPKG rtn to set the cursor position */
5418 1 SCR$ERASE_PAGE ENTRY (ANY VALUE, ANY VALUE), /* SCRPKG rtn to home the cursor & clear the entire screen */
5419 1 SCR$UP_SCROLL ENTRY, /* SCRPKG rtn to scroll up one line */
5420 1 SCR$STOP_OUTPUT ENTRY; /* Rtn to stop SCRPKG output stream */
5421 1
5422 1 Declare
5423 1 FIN_SEQ GLOBALREF, /* Finish escape sequence for display terminal */
5424 1 L FIXED BINARY(7), /* Length */
5425 1 S CHAR(1), /* First character of string */
5426 1
5427 1 BOT_CURS GLOBALREF, /* Place cursor on bottom of screen */
5428 1 L FIXED BINARY(7), /* Length */
5429 1 S CHAR(1), /* First character of string */
5430 1
5431 1 DFSPEC CHAR(8) BASED; /* Dummy display file spec descriptor */
5432 1
5433 1 M->MRBSV DIS CL REQ = NO; /* Indicate display cleanup is no longer required */
5434 1 CALL LIB$SET_BUFFER(0); /* Indicate "clear buffer mode" to SCRPKG */
5435 1 /* ... and output what's left in the buffer */
```

EXECUTE_REQUEST
V04-000

J 10
16-SEP-1984 02:15:40 VAX-11 PL/I X2.1-273 Page 58
5-SEP-1984 15:10:53 ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PLI;1 (45)

```
5436 1 CALL PUT TO SCREEN(FIN_SEQ.L,FIN_SEQ.S);
5437 1 IF DISPLAYING = YES
5438 1 THEN DO:
5439 2 DISPLAYING = NO;
5440 2 CALL SCR$SET_CURSOR(24,1);
5441 2 CALL SCR$UP_SCROLL();
5442 2 END;
5443 1 CALL SCR$STOP_OUTPUT();
5444 1 RETURN(NORMAL);
5445 1 END DISPLAY_CLEANUP;
5446
```

```
/* Call SCRPKG for finish sequence, */
/* If actual output has begun, */

/* Indicate display output has stopped, */
/* ... place cursor on bottom line, */
/* ... and scroll up one line */

/* Stop output stream if present */
/* Return */
```



```
5447 INPUT_INIT: Procedure Returns(Fixed Binary(31));
5448
5449 /*
5450 /*++
5451 /*
5452 /* FUNCTIONAL DESCRIPTION:
5453 /*
5454 /* INPUT_INIT
5455 /*
5456 /* Called by REQUEST_INIT or MFSUM_REQUEST to open the input
5457 /* (playback) file, performing various sanity checks on it.
5458 /*
5459 /* IMPLICIT INPUTS:
5460 /*
5461 /* MRBPTR (or M) points to active MRB. In particular, MRB$A_INPUT
5462 /* points to string descriptor of file-spec to be opened.
5463 /*
5464 /* IMPLICIT OUTPUTS:
5465 /*
5466 /* Input file has been opened.
5467 /* H and MCAS$A_INPUT_PTR both point to first data byte of header record.
5468 /* MCAS$L_INPUT_LEN contains length of header record.
5469 /*
5470 /* ROUTINE VALUE:
5471 /*
5472 /* SSS$NORMAL, or failing MONITOR status code.
5473 /*
5474 /* SIDE EFFECTS:
5475 /*
5476 /* /INPUT file (INPUT_FILE) is positioned to the file header record.
5477 /*
5478 /*--
5479 /*/
5480
5481 /*
5482 /*
5483 /*
5484 /* INCLUDE FILES
5485 /*
5486 /*
5487 /*/
5488
5489 %INCLUDE MONDEF; /* Monitor utility structure definitions */
6257
6258 /*
6259 /*
6260 /*
6261 /* MESSAGE DEFINITIONS
6262 /*
6263 /*
6264 /*/
6265
6266 Declare
6267 MNR$_PREMEOF FIXED BINARY(31) GLOBALREF VALUE,
6268 MNR$_INVINPFIL FIXED BINARY(31) GLOBALREF VALUE,
6269 MNR$_UNSTLEV FIXED BINARY(31) GLOBALREF VALUE;
```

```
6270 1
6271 1  /*
6272 1  /*
6273 1  /*
6274 1  /*
6275 1  /*
6276 1  /*
6277 1  /*/
6278
6279 Declare
6280 MON_ERR ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE), /* MONITOR MACRO-32 routine to log synchronous error */
6281 MAX_REC_SIZE FIXED BINARY(31) GLOBALREF VALUE, /* Max record size for PLAYBACK & RECORD files */
6282 NORMAL FIXED BINARY(31) GLOBALREF, /* MONITOR normal status value */
6283 MRBPTR POINTER GLOBALREF, /* Pointer to MRB (Monitor Request Block) */
6284 M POINTER DEFINED(MRBPTR), /* Synonym for MRBPTR */
6285 MCAPTR POINTER GLOBALREF, /* Pointer to MCA (Monitor Communication Area) */
6286 MC POINTER DEFINED(MCAPTR), /* Synonym for MCAPTR */
6287 H POINTER GLOBALREF, /* Pointer to input file header */
6288 ST_LEVEL_CUR CHAR(8) GLOBALREF, /* Current MONITOR recording file structure level */
6289 ST_LEVEL_PB CHAR(8) GLOBALDEF, /* MONITOR recording file structure level from input file */
6290 NEXT_REC FIXED BINARY(31) GLOBALREF VALUE, /* Read next record indicator for READ INPUT rtn */
6291 HEADER_TYPE FIXED BINARY(15) GLOBALREF, /* Type for MONITOR recording file header */
6292 INPUT_CPTR POINTER GLOBALREF, /* Ptr to input buffer count word */
6293 INPUT_DATA CHAR(MAX_REC_SIZE) VARYING BASED(INPUT_CPTR); /* Playback file input buffer */
6294
6295 Declare
6296 INPUT_FILE FILE RECORD INPUT; /* Monitor Input (Playback) File */
6297
6298  /*
6299  /*
6300  /*
6301  /*
6302  /*
6303  /*
6304  /*/
6305
6306 Declare
6307
6308 TEMP_TYPE BIT(8) ALIGNED, /* Temporary area for record type byte */
6309 TEMP_PTR POINTER,
6310 01 TEMP BASED(TEMP_PTR),
6311 02 L FIXED BINARY(15),
6312 02 DC FIXED BINARY(15),
6313 02 A POINTER,
6314 TEMP_STR CHAR(TEMP.L) BASED(TEMP.A);
6315
6316 Declare
6317 TEMP_INPUT_PTR FIXED BINARY(31) BASED(ADDR(MC->MCA$A_INPUT_PTR)); /* Alias for MCA$A_INPUT_PTR for computation */
6318
6319
6320 M->MRBSV INP CL REQ = YES; /* Indicate input cleanup is required */
6321 CLOSE FILE(INPUT_FILE); /* Make sure file is closed before opening */
6322 INPUT_CPTR = NULL(); /* Indicate no input buffer yet */
6323 TEMP_PTR = M->MRBSA INPUT; /* Set up ptr to input file name string */
6324 OPEN_FILE(INPUT_FILE) TITLE(TEMP_STR) /* Open the input recording file for playback */
6325 ENVIRONMENT(_SHARED_WRITE); /* and shared read (but have to use SHARED_WRITE) */
```



```
6326 1      ALLOCATE INPUT DATA;                                /* Allocate space for input buffer (for life of request) */
6327 1      MC->MCASA_INPUT_PTR = INPUT_CPTR;                    /* Get ptr to first byte of input buffer */
6328 1      TEMP_INPUT_PTR = TEMP_INPUT_PTR + 2;                 /* Advance ptr beyond length word */
6329 1      CALL READ_INPUT(NEXT_REC);                             /* Read first (file header) record */
6330 1      IF MC->MCASV_EOF                                     /* If end-of-file, */
6331 1      THEN DO;
6332 2          CALL MON_ERR(MNRS_PREMEOF);                        /* Can't find file header; log the error */
6333 2          RETURN (MNRS_PREMEOF);                             /* ... and return to caller */
6334 2      END;
6335 1
6336 1      H = MC->MCASA_INPUT_PTR;                                /* Establish ptr to file header */
6337 1      TEMP_TYPE = UNSPEC(HEADER_TYPE);                      /* Get header type into a byte for compare */
6338 1      IF H->MNR_HDR$B_TYPE ^= TEMP_TYPE;                    /* If first record is not a file header or ... */
6339 1      SUBSTR(H->MNR_HDR$T_LEVEL,1,3) ^= SUBSTR(ST_LEVEL_CUR,1,3) /* ... MONITOR ID is not OK, */
6340 1      THEN DO;
6341 2          CALL MON_ERR(MNRS_INVINPFIL);                        /* Log an error */
6342 2          RETURN(MNRS_INVINPFIL);                             /* ... and return to caller */
6343 2      END;
6344 1
6345 1      IF SUBSTR(H->MNR_HDR$T_LEVEL,7,2) ^= SUBSTR(ST_LEVEL_CUR,7,2) /* If format level is not OK, */
6346 1      THEN DO;
6347 2          CALL MON_ERR(MNRS_UNSTLEV);                          /* Log an error */
6348 2          RETURN(MNRS_UNSTLEV);                               /* ... and return to caller */
6349 2      END;
6350 1
6351 1      ST_LEVEL_PB = H->MNR_HDR$T_LEVEL;                       /* Save playback structure level */
6352 1
6353 1      RETURN(NORMAL);
6354 1      END INPUT_INIT;
6355
```

```
6356 DISP_TEMPLATE: Procedure (DCDB, OUTPUT_IND)
6357 Returns(Fixed-Binary(31));
6358
6359 /*
6360 /*++
6361 /*
6362 /* FUNCTIONAL DESCRIPTION:
6363 /*
6364 /* DISP_TEMPLATE
6365 /*
6366 /* Called by DISPLAY_EVENT, SUMMARY_EVENT and PUT_SUMM_PAGE to
6367 /* form and write a template for the indicated class. The template
6368 /* consists of everything on the screen except actual data. This
6369 /* includes the first 7 lines of the screen, the footing line and
6370 /* the line item identifiers. If a bar graph has been requested,
6371 /* the graph box is also included.
6372 /*
6373 /* INPUTS:
6374 /*
6375 /* DCDB -- Pointer to the CDB (Class Descriptor Block)
6376 /* of the class to be displayed.
6377 /*
6378 /* OUTPUT_IND -- Output indicator bit. If set, DISP_TEMPLATE
6379 /* sends terminal display commands to the SCRPKG
6380 /* and requests it to output to the screen. If
6381 /* OUTPUT_IND is not set, DISP_TEMPLATE sends
6382 /* terminal display commands to the SCRPKG, but
6383 /* does not request immediate screen output.
6384 /*
6385 /* OUTPUTS:
6386 /*
6387 /* None
6388 /*
6389 /* ROUTINE VALUE:
6390 /*
6391 /* $$$_NORMAL, or failing MONITOR status code.
6392 /*
6393 /*--
6394 /*/
6395
```



```
6396 | 1 | /*
6397 | 1 | /*
6398 | 1 | /*
6399 | 1 | /*
6400 | 1 | /*
6401 | 1 | /*
6402 | 1 | /*
6403 | 1 | /*
6404 | 1 | %INCLUDE MONDEF; /* Monitor utility structure definitions */
7172 | 1 |
7173 | 1 | /*
7174 | 1 | /*
7175 | 1 | /*
7176 | 1 | /*
7177 | 1 | /*
7178 | 1 | /*
7179 | 1 | /*
7180 | 1 |
7181 | 1 | Declare
7182 | 1 | MON_ERR ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE), /* MONITOR MACRO-32 routine to log synchronous error
7183 | 1 | TEMPLATE ENTRY(POINTER VALUE) /* BLISS rtn to output template */
7184 | 1 | RETURNS(FIXED BINARY(31)),
7185 | 1 | SCR$SET_CURSOR ENTRY (ANY VALUE, ANY VALUE), /* SCRPKG rtn to set the cursor position */
7186 | 1 | DISPLAY_PUT ENTRY(ANY, FIXED BINARY(31), ANY, ANY) /* MACRO-32 rtn to put a display string */
7187 | 1 | OPTIONS(VARIABLE)
7188 | 1 | RETURNS(FIXED BINARY(31));
7189 | 1 |
7190 | 1 | /*
7191 | 1 | /*
7192 | 1 | /*
7193 | 1 | /*
7194 | 1 | /*
7195 | 1 | /*
7196 | 1 | /*
7197 | 1 |
7198 | 1 | Declare
7199 | 1 | MNR$_DISPERR FIXED BINARY(31) GLOBALREF VALUE;
7200 | 1 |
```

```
7201 1
7202 1  /*
7203 1  /*
7204 1  /*
7205 1  /*
7206 1  /*
7207 1  /*
7208 1  /*
7209 1  /*
7210 1  %REPLACE      NOT_SUCCESSFUL      BY '0'B;      /* Failing status bit */
7211 1  %REPLACE      YES                  BY '1'B;      /* For general use */
7212 1  %REPLACE      NO                   BY '0'B;      /* For general use */
7213 1
7214 1  /*
7215 1  /*
7216 1  /*
7217 1  /*
7218 1  /*
7219 1  /*
7220 1  /*
7221 1
7222 1  Declare
7223 1  VTWIDTH      FIXED BINARY(31) GLOBALREF VALUE;      /* Width of video terminal */
7224 1  VTHEIGHT    FIXED BINARY(31) GLOBALREF VALUE;      /* Height of video terminal */
7225 1
7226 1  Declare
7227 1  CDBPTR      POINTER GLOBALREF,      /* Pointer to CDB (Class Descriptor Block) */
7228 1  C            POINTER DEFINED(CDBPTR),      /* Synonym for CDBPTR */
7229 1  MRBPTR     POINTER GLOBALREF,      /* Pointer to MRB (Monitor Request Block) */
7230 1  M           POINTER DEFINED(MRBPTR),      /* Synonym for MRBPTR */
7231 1  MCAPTR     POINTER GLOBALREF,      /* Pointer to MCA (Monitor Communication Area) */
7232 1  MC          POINTER DEFINED(MCAPTR),      /* Synonym for MCAPTR */
7233 1  SPTR       POINTER GLOBALREF;      /* Pointer to SYI (System Information Area) */
7234 1
7235 1  Declare
7236 1  NORMAL      FIXED BINARY(31) GLOBALREF;      /* MONITOR normal return status */
7237 1
7238 1  Declare
7239 1  INP_COMM_STR CHAR(MNR_HDR$K_MAXCOMLEN) GLOBALREF,      /* User comment string from input file */
7240 1  INP_COMM_LEN FIXED BINARY(15) GLOBALREF;      /* Actual length of comment string */
7241 1
```



```
7242 1 Declare
7243 1 1 ANNCE_STR GLOBALREF,
7244 1 2 L FIXED BINARY(7),
7245 1 2 S CHAR(1);
7246 1
7247 1 Declare
7248 1 1 STATUS_STR GLOBALREF,
7249 1 2 L FIXED BINARY(7),
7250 1 2 S CHAR(1),
7251 1 STATUS_PARM CHAR(12) GLOBALREF;
7252 1
7253 1 Declare
7254 1 1 TABHEAD_STR GLOBALREF,
7255 1 2 L FIXED BINARY(7),
7256 1 2 S CHAR(1),
7257 1 TABHEAD_PARM POINTER STATIC,
7258 1 PCENT_STR CHAR(2) GLOBALREF,
7259 1 BLANK_STR CHAR(2) GLOBALREF;
7260 1
7261 1 Declare
7262 1 1 PROCHEAD_STR GLOBALREF,
7263 1 2 L FIXED BINARY(7),
7264 1 2 S CHAR(1);
7265 1
7266 1 Declare
7267 1 1 MF_STATHEAD_STR GLOBALREF,
7268 1 2 L FIXED BINARY(7),
7269 1 2 S CHAR(1);
7270 1
7271 1
```

/* Announcement FAO control string */
/* Length */
/* First character of string */

/* Status FAO control string */
/* Length */
/* First character of string */
/* 3 longword FAOL parms for status display */

/* Tabular heading control string */
/* Length */
/* First character of string */
/* FAOL parm indicating % or blank */
/* Percent symbol cstring */
/* Blank character cstring */

/* PROCESSES heading control string */
/* Length */
/* First character of string */

/* M.F. summary statistic heading control string */
/* Length */
/* First character of string */

```
7272 | 1 /*
7273 | 1 /*
7274 | 1 /*
7275 | 1 /*
7276 | 1 /*
7277 | 1 /*
7278 | 1 /*/
7279 | 1
7280 | 1 Declare
7281 | 1 CALL          FIXED BINARY(31) STATIC,
7282 | 1 STATUS       BIT(1)  BASED(ADDR(CALL));
7283 | 1
7284 | 1 Declare
7285 | 1 1 DPUT_FLAGS,
7286 | 1 2 FAOL_REQUESTED BIT(8) ALIGNED,
7287 | 1 2 OUTP_REQUESTED BIT(8) ALIGNED,
7288 | 1 PUT_LEN        FIXED BINARY(31);
7289 | 1
7290 | 1 Declare
7291 | 1 DCDB          POINTER,
7292 | 1 OUTPUT_IND    BIT(1) ALIGNED,
7293 | 1 I             FIXED BINARY(15);
7294 | 1
7295 | 1 Declare
7296 | 1 SPEC_SYSTEM_SCREEN BIT(1) ALIGNED;
7297 | 1
7298 | 1 Declare
7299 | 1 1 TITLE_PARS  STATIC,
7300 | 1 2 BLANKS      FIXED BINARY(31),
7301 | 1 2 TITLE_PTR   POINTER,
7302 | 1 2 PCENT_WID   FIXED BINARY(31),
7303 | 1 2 NODE_PTR    POINTER,
7304 | 1 TITLE_LEN     FIXED BINARY(7) BASED(TITLE_PTR),
7305 | 1 NODE_LEN      FIXED BINARY(7) BASED(NODE_PTR),
7306 | 1 1 TITLE_STR   GLOBALREF,
7307 | 1 2 L           FIXED BINARY(7),
7308 | 1 2 S           CHAR(1);
7309 | 1
7310 | 1 Declare
7311 | 1 1 COMM_PARS    STATIC,
7312 | 1 2 BLANKS      FIXED BINARY(31),
7313 | 1 2 COMM_LEN     FIXED BINARY(31),
7314 | 1 2 COMM_ADDR    POINTER,
7315 | 1 1 COMM_STR     GLOBALREF,
7316 | 1 2 L           FIXED BINARY(7),
7317 | 1 2 S           CHAR(1);
7318 | 1
7319 | 1 Declare
7320 | 1 1 SYS_HEAD_PARS STATIC,
7321 | 1 2 SYS_NODE_PTR  POINTER,
7322 | 1 2 STATLONG_LEN  FIXED BINARY(31) INIT(7),
7323 | 1 2 STATLONG_ADDR POINTER,
7324 | 1
7325 | 1 1 SYS_HEAD_STR  GLOBALREF,
7326 | 1 2 L           FIXED BINARY(7),
7327 | 1 2 S           CHAR(1),
```

OWN STORAGE

```
/* Holds function value (return status) of called ro
/* Low-order status bit for called routines */

/* DISPLAY_PUT routine flags */
/* YES => Xlate buffer with FAOL first */
/* YES => Really output buffer */
/* Length of buffer for DISPLAY_PUT to put */

/* Pointer to current display class CDB */
/* YES => output the template */
/* Index for DO loop */

/* YES => special screen for SYSTEM class */

/* FAOL parms for title display line */
/* Number of preceding blanks */
/* Pointer to title cstring */
/* Width of percent string (0 or 4) */
/* Pointer to DECnet node name cstring */
/* Length of title string */
/* Length byte of node name cstring */
/* Title FAOL control string */
/* Length */
/* First character of string */

/* FAOL parms for comment display line */
/* Number of preceding blanks */
/* Length of comment */
/* Address of comment string */
/* Comment FAOL control string */
/* Length */
/* First character of string */

/* FAOL parms for SYSTEM heading line */
/* Pointer to DECnet node name cstring */
/* Length of requested stat */
/* Addr of requested stat */

/* SYSTEM class heading control string */
/* Length */
/* First character of string */
```



```
7328 1
7329 1 STAT LONG (4) CHAR(7) GLOBALREF, /* Table of 7-char statistic strings */
7330 1 SYS_BOX_STR_ADDR POINTER GLOBALREF, /* Address of screen box string */
7331 1 SYS_BOX_STR CHAR(1) BASED(SYS_BOX_STR_ADDR), /* SYSTEM screen boxes (1st char) */
7332 1 SYS_BOX_STR_LEN FIXED BINARY(15) GLOBALREF, /* Its length */
7333 1 SYS_TEXT_STR CHAR(1) GLOBALREF, /* SYSTEM screen text */
7334 1 SYS_TEXT_STR_LEN FIXED BINARY(15) GLOBALREF, /* Its length */
7335 1 SYS_FAO_STR CHAR(1) GLOBALREF, /* SYSTEM FAO string */
7336 1 SYS_FAO_STR_LEN FIXED BINARY(15) GLOBALREF; /* Its length */
7337 1
7338 1 Declare
7339 1 SYS_BOX_PARMS, /* FAOL parms for SYSTEM screen boxes */
7340 1 2 SBP1 FIXED BINARY(31), /* Free List bar range value */
7341 1 2 SBP2 POINTER, /* Pointer to 'K' or null cstring */
7342 1 2 SBP3 FIXED BINARY(31), /* Mod List bar range value */
7343 1 2 SBP4 POINTER; /* Pointer to 'K' or null cstring */
7344 1
7345 1 Declare
7346 1 BU_SYS_SINGLE GLOBALREF, /* Bar graph range values for SYSTEM class (single s
7347 1 2 BSS_RANGE (1:17) FIXED BINARY(31),
7348 1 K_STR CHAR(2) GLOBALREF, /* K symbol for box */
7349 1 NULL_STR FIXED BINARY(15) INIT(0); /* Dummy null symbol for box */
7350 1
7351 1
```

```
7352 1 MC->MCASV_ERA_SCRL = NO; /* Indicate no need to erase scrolling region ... */
7353 1 /* ... (display area) for PROCESSES and homogs */
7354 1 IF DCDB->CDBSV_HOMOG
7355 1 THEN DCDB->CDBSA_CDX->CDX$SL_PREV_DCT = 0; /* Init count of previous display elements ... */
7356 1 /* ... for homogeneous class */
7357 1
7358 1 IF DCDB->CDBSV_SYSCLS & DCDB->CDBSB_ST ^= ALL_STAT /* If special SYSTEM screen, */
7359 1 THEN SPEC_SYSTEM_SCREEN = YES; /* Set a bit for quick reference */
7360 1 ELSE SPEC_SYSTEM_SCREEN = NO; /* Otherwise, turn it off */
7361 1
7362 1
7363 1 /*
7364 1 /* Send announcement string to SCRPKG via DISPLAY_PUT routine
7365 1 /* This string is independent of screen style.
7366 1 /*/
7367 1
7368 1 PUT_LEN = ANNCE_STR.L; /* Get length of this put */
7369 1 FAOL_REQUESTED = NO; /* No need to go thru $FAOL */
7370 1 OUTP_REQUESTED = NO; /* Not ready to actually output it yet */
7371 1 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,ANNCE_STR.S);
7372 1 /* Send announcement string to SCRPKG */
7373 1 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
7374 1
7375 1 /*
7376 1 /* Send status (footing) string to SCRPKG via DISPLAY_PUT
7377 1 /* routine. This string is independent of screen style.
7378 1 /* Skip it, however, for multi-file summary.
7379 1 /*/
7380 1
7381 1 IF M->MRBSV_MFSUM = NO /* If not multi-file summary, */
7382 1 THEN DO:
7383 2 PUT_LEN = STATUS_STR.L; /* Get length of this put */
7384 2 FAOL_REQUESTED = YES; /* Request a run thru $FAOL */
7385 2 OUTP_REQUESTED = NO; /* Not ready to actually output it yet */
7386 2 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,STATUS_STR.S,STATUS_PARM$);
7387 2 /* Send status string to SCRPKG */
7388 2 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
7389 2 END;
7390 1
```



```
7391 1
7392 1 /*
7393 1 /* Send title string to SCRPKG via DISPLAY_PUT routine.
7394 1 /* Includes DECnet node name if one is present
7395 1 /* This string is independent of screen style.
7396 1 /*/
7397 1
7398 1 TITLE_PTR = DCDB->CDB$A TITLE; /* Establish title pointer */
7399 1 TITLE_PARAMS.BLANKS = DIVIDE((VTWIDTH - TITLE_LEN),2,31) - 1; /* Compute preceding blanks */
7400 1 IF DCDB->CDB$V_PERCENT = YES & M->MRB$V_MFSUM = NO /* If percent requested for other than m.f. summary, */
7401 1 THEN PCENT_WID = 4; /* then put out % string */
7402 1 ELSE PCENT_WID = 0; /* else don't put % string */
7403 1 PUT_LEN = TITLE_STR.L; /* Get length of this put */
7404 1 NODE_PTR = ADDR(SPTR->MNR_SYIST_NODENAME); /* Set up ptr to node name cstring */
7405 1 IF NODE_LEN = 0 ! SPEC_SYSTEM_SCREEN /* If node name non-existent, or special SYSTEM screen, */
7406 1 THEN PUT_LEN = PUT_LEN - T6; /* then chop off node name line */
7407 1 FAOL_REQUESTED = YES; /* Request a run thru $FAOL */
7408 1 OUTP_REQUESTED = NO; /* Not ready to actually output it yet */
7409 1 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,TITLE_STR.S,TITLE_PARAMS); /* Send title line to SCRPKG */
7410 1 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
7411 1
7412 1 /*
7413 1 /* If special screen display for SYSTEM class,
7414 1 /* send a heading string including DECnet node
7415 1 /* name and requested statistic.
7416 1 /*/
7417 1
7418 1 IF SPEC_SYSTEM_SCREEN
7419 1 THEN DO;
7420 2 STATLONG_ADDR = ADDR(STAT_LONG(DCDB->CDB$B_ST)); /* Get addr of correct stat string */
7421 2 SYS_NODE_PTR = NODE_PTR; /* Get address of node name cstring */
7422 2 PUT_LEN = SYS_HEAD_STR.L; /* Get length of this put */
7423 2 FAOL_REQUESTED = YES; /* Request a run thru $FAOL */
7424 2 OUTP_REQUESTED = NO; /* Not ready to actually output it yet */
7425 2 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SYS_HEAD_STR.S,SYS_HEAD_PARAMS); /* Send SYSTEM heading to SCRPKG */
7426 2 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
7427 2 END;
7428 2
7429 2
7430 1
7431 1 /*
7432 1 /* Send user's comment string to SCRPKG via DISPLAY_PUT routine.
7433 1 /* This string is independent of screen style.
7434 1 /*/
7435 1
7436 1 IF M->MRB$V_MFSUM = NO & INP_COMM_LEN ^= 0 /* If not m.f. summary and an input comment exists, */
7437 1 & SPEC_SYSTEM_SCREEN = NO /* and not the special SYSTEM screen, */
7438 1 THEN DO;
7439 2 COMM_LEN = INP_COMM_LEN; /* Move length to parm list */
7440 2 COMM_ADDR = ADDR(INP_COMM_STR); /* Move address to parm list */
7441 2 COMM_PARAMS.BLANKS = DIVIDE((VTWIDTH - COMM_LEN),2,31) - 1; /* Compute preceding blanks */
7442 2 PUT_LEN = COMM_STR.L; /* Get length of this put */
7443 2 FAOL_REQUESTED = YES; /* Request a run thru $FAOL */
7444 2 OUTP_REQUESTED = NO; /* Not ready to actually output it yet */
7445 2 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,COMM_STR.S,COMM_PARAMS); /* Send comment line to SCRPKG */
7446 2
```

```
IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
END;
```



```
7450 : 1 /*
7451 : 1 /* For standard classes, call TEMPLATE to put item
7452 : 1 /* names and build FAO string for actual data for
7453 : 1 /* tabular or bar-style screen. Skip, however, for
7454 : 1 /* the special SYSTEM display
7455 : 1 /*/
7456 : 1
7457 : 1 IF DCDB->CDB$V_STD & SPEC_SYSTEM_SCREEN = NO /* If standard class, and not SYSTEM screen, */
7458 : 1 THEN DO; /* Put item names and build FAO string */
7459 : 2 CALL = TEMPLATE(DCDB); /* Check status */
7460 : 2 IF STATUS = NOT_SUCCESSFUL
7461 : 2 THEN DO;
7462 : 3 CALL MON_ERR(MNR$ DISPERR,CALL); /* Log the error */
7463 : 3 RETURN(MNR$ DISPERR); /* ... and return with status */
7464 : 3 END;
7465 : 2 END;
7466 : 1
7467 : 1 /*
7468 : 1 /* Send heading string (and box, if bar graph)
7469 : 1 /* to SCRPKG via DISPLAY_PUT routine.
7470 : 1 /*/
7471 : 1
7472 : 1 IF M->MRB$V_MFSUM = NO & SPEC_SYSTEM_SCREEN = NO /* Only do it if not multi-file summary and not spec
7473 : 1 THEN
7474 : 1
7475 : 1 /*
7476 : 1 /* Put PROCESSES Heading
7477 : 1 /*/
7478 : 1
7479 : 1 IF ^ DCDB->CDB$V_STD & DCDB->CDB$B_ST = REG_PROC /* Put out regular PROCESSES heading */
7480 : 1 THEN DO;
7481 : 2 PUT_LEN = PROCHEAD_STR.L; /* Length of put */
7482 : 2 FAO_REQUESTED = NO; /* No $FAOL required */
7483 : 2 OUTP_REQUESTED = OUTPUT_IND; /* Output it if caller requested */
7484 : 2 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,PROCHEAD_STR.S.); /* Hand heading over to SCRPKG */
7485 : 2 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
7486 : 2 END;
7487 : 1
7488 : 1 /*
7489 : 1 /* Put Tabular Heading
7490 : 1 /*/
7491 : 1
7492 : 1 ELSE IF DCDB->CDB$V_STD & DCDB->CDB$B_ST = ALL_STAT /* All statistics requested for STD class? */
7493 : 1 THEN DO; /* Tabular display */
7494 : 2 IF DCDB->CDB$V_WIDE /* If a wide display (for DISK), */
7495 : 2 THEN CALL SCR$SET_CURSOR(6,44); /* ... then set appropriate cursor */
7496 : 2 ELSE CALL SCR$SET_CURSOR(6,40); /* ... else set it to the usual place */
7497 : 2 IF DCDB->CDB$V_PERCENT
7498 : 2 THEN TABHEAD_PARM = ADDR(PCENT_STR); /* Include % symbol in heading */
7499 : 2 ELSE TABHEAD_PARM = ADDR(BLANK_STR); /* Exclude % symbol from heading */
7500 : 2 PUT_LEN = TABHEAD_STR.L; /* Length of put */
7501 : 2 FAO_REQUESTED = YES; /* Request a run thru $FAOL */
7502 : 2 OUTP_REQUESTED = OUTPUT_IND; /* Output it if caller requested */
7503 : 2 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,TABHEAD_STR.S,TABHEAD_PARM); /* Hand heading over to SCRPKG */
7504 : 2 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
7505 : 2
```

EXECUTE REQUEST
V04-000

7506	2
7507	1

END:

K 11
16-SEP-1984 02:15:46
5-SEP-1984 15:10:53

VAX-11 PL/I X2.1-273 Page 72
ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PL1;1 (55)

EXE
V04

[illegible]


```
7508 1 1 /*
7509 1 1 /* Put Bar Graph Heading
7510 1 1 /*
7511 1 1
7512 1 1
7513 2 1 ELSE BEGIN;
7514 2 2 Declare
7515 2 2 CURGR_VAL FIXED BINARY(31),
7516 2 2 MAXGR_VAL FIXED BINARY(31),
7517 2 2 GR_INCR FIXED BINARY(31),
7518 2 2 RANGE FIXED BINARY(31),
7519 2 2 CHAR_ADDR POINTER;
7520 2 2
7521 2 2 Declare
7522 2 2 1 BARHEAD_STR GLOBALREF,
7523 2 2 2 L FIXED BINARY(7),
7524 2 2 2 S CHAR(1),
7525 2 2 1 BARHEAD_PARMs,
7526 2 2 2 BP1 FIXED BINARY(31),
7527 2 2 2 BP2 POINTER,
7528 2 2 2 BP3 FIXED BINARY(31),
7529 2 2 2 BP4 POINTER,
7530 2 2 2 BP5 FIXED BINARY(31),
7531 2 2 2 BP6 POINTER,
7532 2 2 2 BP7 FIXED BINARY(31),
7533 2 2 2 BP8 POINTER,
7534 2 2 2 BP9 FIXED BINARY(31),
7535 2 2 2 BP10 FIXED BINARY(31),
7536 2 2 2 BP11 POINTER;
7537 2 2
7538 2 2 Declare
7539 2 2 1 STATHEAD_STR GLOBALREF,
7540 2 2 2 L FIXED BINARY(7),
7541 2 2 2 S CHAR(1),
7542 2 2 1 STATHEAD_PARMs,
7543 2 2 2 L FIXED BINARY(31) INIT(3),
7544 2 2 2 A POINTER,
7545 2 2 STAT_HEAD (4) CHAR(3) GLOBALREF;
```

/* Bar graph display */

/* Current graph value (for heading) */
/* Max (right-edge) graph value for heading */
/* Increment value for heading */
/* Range for heading values */
/* Addr of symbol char for heading *//* Bar graph heading control string */
/* Length */
/* First character of string */
/* FAOL parms for graph heading line */
/* Graph heading value */
/* Graph heading symbol string ptr */
/* Graph heading value */
/* Graph heading symbol string ptr */
/* Graph heading value */
/* Graph heading symbol string ptr */
/* Graph heading value */
/* Graph heading symbol string ptr */
/* 1=> advance heading one byte to right */
/* Graph heading value */
/* Graph heading symbol string ptr *//* Bar graph statistic heading control string */
/* Length */
/* First character of string */
/* FAOL parms for statistic heading */
/* Statistic heading string length */
/* Pointer to heading string */
/* Table of 3-char heading strings */

```
7546      CALL = PUT_BOX();
7547      IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);
7548
7549      /*
7550      /* Put heading line on top of the box.
7551      /*
7552
7553      IF DCDB->CDB$V_PERCENT
7554      THEN DO;
7555          CHAR_ADDR = ADDR(PCENT_STR);
7556          CURGR_VAL = 0;
7557          RANGE = 100;
7558          END;
7559      ELSE IF DCDB->CDB$V_KUNITS
7560      THEN DO;
7561          CHAR_ADDR = ADDR(K_STR);
7562          CURGR_VAL = DIVIDE(DCDB->CDB$L_MIN,1000,31); /* Compute first value */
7563          RANGE = DIVIDE(DCDB->CDB$L_RANGE,1000,31); /* ... and range */
7564          END;
7565      ELSE DO;
7566          CHAR_ADDR = ADDR(NULL_STR);
7567          CURGR_VAL = DCDB->CDB$L_MIN;
7568          RANGE = DCDB->CDB$L_RANGE;
7569          END;
7570      GR_INCR = DIVIDE(RANGE,4,31);
7571      MAXGR_VAL = CURGR_VAL + RANGE;
7572      BP1 = CURGR_VAL;
7573      BP2 = CHAR_ADDR;
7574      CURGR_VAL = CURGR_VAL + GR_INCR;
7575      BP3 = CURGR_VAL;
7576      BP4 = CHAR_ADDR;
7577      CURGR_VAL = CURGR_VAL + GR_INCR;
7578      BP5 = CURGR_VAL;
7579      BP6 = CHAR_ADDR;
7580      CURGR_VAL = CURGR_VAL + GR_INCR;
7581      BP7 = CURGR_VAL;
7582      BP8 = CHAR_ADDR;
7583      CURGR_VAL = CURGR_VAL + GR_INCR;
7584      IF DCDB->CDB$V_PERCENT : DCDB->CDB$V_KUNITS
7585      THEN BP9 = 0;
7586      ELSE BP9 = 1;
7587
7588      BP10 = MAXGR_VAL;
7589      BP11 = CHAR_ADDR;
```

```
/* Put larger bar graph box to SCRPKG */
/* Check status */

/* Heading values are percents */
/* Use % symbol for heading */
/* First value is 0 */
/* Range is 100 */

/* Values in units of 1000 */
/* Use K symbol for heading */
/* Compute first value */
/* ... and range */

/* Heading values are as is */
/* Use no (null) symbol for heading */
/* Compute first value */
/* ... and range */

/* Compute increment between values */
/* ... and max (right-most) value */
/* Fill in FAOL parms to put heading */
/* ..... */
/* Compute next value */
/* ..... */
/* ..... */
/* Compute next value */
/* ..... */
/* ..... */
/* Compute next value */
/* ..... */
/* ..... */
/* Compute next value */
/* If units symbol is printable, */
/* ... then do not advance one space */
/* ... else advance a space, so last value */
/* ... is on right edge of box */
/* Next parm is the last value */
/* ... addr of units symbol */
```


[illegible]

```
7616 1 ELSE
7617 1 IF SPEC_SYSTEM_SCREEN
7618 1 THEN
7619 1 DO;
7620 2 DCDB->CDB$A_FAOCTR = ADDR(SYS_FAO_STR);
7621 2 DCDB->CDB$L_FAOCTR = SYS_FAO_STR_LEN;
7622 2 IF BSS_RANGE(14) >= 10000
7623 2 THEN DO;
7624 3 SBP1 = DIVIDE(BSS_RANGE(14),1000,31);
7625 3 SBP2 = ADDR(K_STR);
7626 3 END;
7627 2 ELSE DO;
7628 3 SBP1 = BSS_RANGE(14);
7629 3 SBP2 = ADDR(NULL_STR);
7630 3 END;
7631 2 IF BSS_RANGE(15) >= 10000
7632 2 THEN DO;
7633 3 SBP3 = DIVIDE(BSS_RANGE(15),1000,31);
7634 3 SBP4 = ADDR(K_STR);
7635 3 END;
7636 2 ELSE DO;
7637 3 SBP3 = BSS_RANGE(15);
7638 3 SBP4 = ADDR(NULL_STR);
7639 3 END;
7640 2 PUT_LEN = SYS_BOX_STR_LEN;
7641 2 FAOL_REQUESTED = YES;
7642 2 OUTP_REQUESTED = NO;
7643 2 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SYS_BOX_STR,SYS_BOX_PARAMS);
7644 2 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);
7645 2 PUT_LEN = SYS_TEXT_STR_LEN;
7646 2 FAOL_REQUESTED = YES;
7647 2 OUTP_REQUESTED = OUTPUT_IND;
7648 2 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SYS_TEXT_STR,);
7649 2 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);
7650 2 END;
7651 1 ELSE
7652 1 DO;
7653 2 PUT_LEN = MF_STATHEAD_STR.L;
7654 2 FAOL_REQUESTED = YES;
7655 2 OUTP_REQUESTED = OUTPUT_IND;
7656 2 CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,MF_STATHEAD_STR.S,);
7657 2 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);
7658 2 END;
7659 1 RETURN(NORMAL);
7660 1
7661 1
7662 1
7663 1
7664 1
7665 1
```

/* At this point, either m.f.summ or spec. SYSTEM sc
/* If special SYSTEM screen, */

/* Get a pre-built FAO control string */
/* ... and its length */
/* If range of Free List bar is large, */
/* then get the number of thousands */
/* and use a 'K' */

/* else use the raw number */
/* and no 'K' */

/* If range of Modified List bar is large, */
/* then get the number of thousands */
/* and use a 'K' */

/* else use the raw number */
/* and no 'K' */

/* Length of put */
/* Request a run thru \$FAOL */
/* Don't output yet */
/* Hand boxes over to SCRPKG */
/* Check status */

/* Length of put */
/* Request a run thru \$FAOL */
/* Output it if caller requested */
/* Output text and display entire screen */
/* Check status */

/* Multi-file summary */

/* Length of put */
/* Request a run thru \$FAOL */
/* Output it if caller requested */
/* Hand statistic heading over to SCRPKG */
/* Check status */

/* Return to caller */

[illegible]

```
7693 2 /*
7694 2 /*
7695 2 /*
7696 2 /*
7697 2 /*
7698 2 /*
7699 2 /*
7700 2
7701 2 Declare
7702 2 FIRST_DATA_LINE FIXED BINARY(31) GLOBALREF VALUE, /* Line number of first data line on screen */
7703 2 LAST_DATA_LINE FIXED BINARY(31) GLOBALREF VALUE, /* Line number of last data line on screen */
7704 2 VTDATALINES FIXED BINARY(31) GLOBALREF VALUE, /* Total data lines on screen */
7705 2 CURSOR_STR CHAR(2) GLOBALREF, /* Cursor control escape sequence */
7706 2 HORIZ_STR CHAR(42) GLOBALREF, /* Horizontal portion of bar graph box */
7707 2 CURROW FIXED BINARY(15), /* Current row counter */
7708 2 CURCOL FIXED BINARY(15); /* Current column counter */
7709 2
7710 2 Declare
7711 2 1 VERT_LINE (5*VTDATALINES), /* Escape string to make vertical lines for box */
7712 2 2 CURSOR CHAR(2), /* Cursor control esc sequence */
7713 2 2 ROW FIXED BINARY(7), /* Row byte */
7714 2 2 COL FIXED BINARY(7), /* Column byte */
7715 2 2 VERT_CHAR CHAR(1); /* Vertical bar character */
7716 2
7717 2 Declare
7718 2 1 HORIZ_LINE (2), /* Escape string to make horiz'l lines for box */
7719 2 2 CURSOR CHAR(2), /* Cursor control esc sequence */
7720 2 2 ROW FIXED BINARY(7), /* Row byte */
7721 2 2 COL FIXED BINARY(7), /* Column byte */
7722 2 2 TOP_BOT CHAR(42); /* Horiz line appearing at top & bot of box */
7723 2
```



```
7724 | 2 /*
7725 | 2 /*      Create and send to the SCRPKG the horizontal (top
7726 | 2 /*      and bottom) lines of the bar graph box.
7727 | 2 /*/
7728 | 2
7729 | 2 HORIZ_LINE.CURSOR(1) = CURSOR_STR; /* Move in cursor control sequence */
7730 | 2 HORIZ_LINE.ROW(1) = FIRST_DATA_LINE - 1; /* Move in row number of top line of box */
7731 | 2 HORIZ_LINE.COL(1) = 38; /* Move in column number */
7732 | 2 TOP BOT(1) = HORIZ_STR; /* Move in the top line */
7733 | 2 HORIZ_LINE.CURSOR(2) = CURSOR_STR; /* Move in cursor control sequence */
7734 | 2 HORIZ_LINE.ROW(2) = LAST_DATA_LINE+1; /* Move in row number of bot line of box */
7735 | 2 HORIZ_LINE.COL(2) = 38; /* Move in column number */
7736 | 2 TOP BOT(2) = HORIZ_STR; /* Move in the bottom line */
7737 | 2 FAOL_REQUESTED = NO; /* FAOL not involved */
7738 | 2 OUTP_REQUESTED = NO; /* ... don't output it yet */
7739 | 2 CALL = DISPLAY_PUT(DPUT_FLAGS,46*2,HORIZ_LINE,); /* Put horizontal lines to SCRPKG */
7740 | 2 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
7741 | 2
7742 | 2 /*
7743 | 2 /*      Now create and send to the SCRPKG the vertical
7744 | 2 /*      lines of the bar graph box.
7745 | 2 /*/
7746 | 2
7747 | 2 I = 0; /* Initialize loop control */
7748 | 2 DO CURROW = FIRST_DATA_LINE TO LAST_DATA_LINE; /* Loop once for each data line in graph */
7749 | 3 DO CURCOL = 38 TO 78 BY 10; /* Loop once for each vert char in a line */
7750 | 4 IF CURCOL = 78 THEN CURCOL = CURCOL + 1; /* Push right-most bar over 1 */
7751 | 4 I = I + 1; /* Update index into VERT_LINE vector */
7752 | 4 VERT_LINE.CURSOR(I) = CURSOR_STR; /* Move in cursor control sequence */
7753 | 4 VERT_LINE.ROW(I) = CURROW; /* Move in row number */
7754 | 4 VERT_LINE.COL(I) = CURCOL; /* Move in column number */
7755 | 4 VERT_CHAR(I) = '|'; /* Move in the vertical bar char */
7756 | 4 END;
7757 | 3 END;
7758 | 2 CALL = DISPLAY_PUT(DPUT_FLAGS,5*5*VTDATA_LINES,VERT_LINE,); /* Put vertical lines to SCRPKG */
7759 | 2 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
7760 | 2
7761 | 2 RETURN(NORMAL); /* Return to caller */
7762 | 2 END PUT_BOX;
7763 | 1
7764 | 1 END DISP_TEMPLATE;
7765 | 1
```

```
7766      COLLECTION_END: Procedure Returns(fixed binary(31));    /* Indicate collection ended */
7767      1
7768      1 /*
7769      1 /*+++
7770      1 /*
7771      1 /* FUNCTIONAL DESCRIPTION:
7772      1 /*
7773      1 /*     COLLECTION_END
7774      1 /*
7775      1 /*     Called by CTRLC, CTRLZ, COLLECTION_EVENT or CLASS_COLLECT whenever it
7776      1 /*     is determined that data collection has reached an end. This can occur
7777      1 /*     when the user strikes CTRL-C or CTRL-Z, an input (playback) file has
7778      1 /*     reached end-of-file, or a requested ending time has occurred.
7779      1 /*
7780      1 /* INPUTS:
7781      1 /*
7782      1 /*     None
7783      1 /*
7784      1 /* OUTPUTS:
7785      1 /*
7786      1 /*     None
7787      1 /*
7788      1 /* IMPLICIT OUTPUTS:
7789      1 /*
7790      1 /*     COLLENDED bit is set.
7791      1 /*
7792      1 /* ROUTINE VALUE:
7793      1 /*
7794      1 /*     SSS_NORMAL
7795      1 /*
7796      1 /* SIDE EFFECTS:
7797      1 /*
7798      1 /*     All timers are canceled, a $WAKE is issued, and the display and
7799      1 /*     "between screens" event flags are set. Also, I/O is canceled on the
7800      1 /*     channel for CTRL-C and CTRL-Z to disable reception of AST's, and
7801      1 /*     on the channel for CTRL-W.
7802      1 /*
7803      1 /*--
7804      1 /*/
7805      1
```



```
7806 1  /*
7807 1  /*
7808 1  /*
7809 1  /*
7810 1  /*
7811 1  /*
7812 1  /*/
7813 1
7814 1 %INCLUDE      SYSSCANTIM;      /* $CANTIM system service */
7821 1 %INCLUDE      SYSS$SETEF;    /* $SETEF system service */
7827 1 %INCLUDE      SYSS$CANCEL;  /* $CANCEL system service */
7833 1 %INCLUDE      MONDEF;        /* Monitor utility structure definitions */
8601 1
8602 1 Declare
8603 1     DISP_EV_FLAG  FIXED BINARY(31) GLOBALREF VALUE, /* Display event flag */
8604 1     BET_EV_FLAG   FIXED BINARY(31) GLOBALREF VALUE, /* 'Between screens' display event flag */
8605 1     HIB_EV_FLAG   FIXED BINARY(31) GLOBALREF VALUE, /* Hibernation event flag */
8606 1     COLLENDED     BIT(1) GLOBALREF, /* YES => collection has ended */
8607 1     MRBPTR        POINTER GLOBALREF, /* Pointer to MRB (Monitor Request Block) */
8608 1     MCAPTR        POINTER GLOBALREF, /* Pointer to MCA (Monitor Communication Area) */
8609 1     CTRLCZ_CHAN   FIXED BINARY(31) GLOBALREF, /* Channel number for CTRL-C and CTRL-Z */
8610 1     CTRLW_CHAN    FIXED BINARY(31) GLOBALREF, /* Channel number for CTRL-W */
8611 1     NORMAL        FIXED BINARY(31) GLOBALREF, /* MONITOR normal status value */
8612 1     CALL          FIXED BINARY(31); /* Holds function value (return status) of called routines */
8613 1
8614 1 CALL = SYSSCANTIM(,); /* Cancel outstanding timer requests */
8615 1 CALL = SYSS$SETEF(HIB_EV_FLAG); /* Wake up if hibernating for future request */
8616 1 CALL = SYSS$SETEF(DISP_EV_FLAG); /* Force final display */
8617 1 CALL = SYSS$SETEF(BET_EV_FLAG); /* Force final screen of multi-screen display */
8618 1 COLLENDED = YES; /* Indicate collection ended */
8619 1 MRBPTR->MRB$Q_ENDING = MCAPTR->MCA$Q_LASTCOLL; /* Establish last collection time as ending */
8620 1 IF CTRLCZ_CHAN ^= 0 THEN CALL = SYSS$CANCEL(CTRLCZ_CHAN); /* Cancel CTRL-C and CTRL-Z handlers */
8621 1 CTRLCZ_CHAN = 0; /* ... and indicate so */
8622 1 IF CTRLW_CHAN ^= 0 THEN CALL = SYSS$CANCEL(CTRLW_CHAN); /* Cancel CTRL-W handler */
8623 1 CTRLW_CHAN = 0; /* ... and indicate so */
8624 1 RETURN(NORMAL); /* Return to caller */
8625 1
8626 1 END COLLECTION_END;
8627 1
```

```
8628      CTRLC: Procedure Returns(fixed binary(31));          /* CTRL-C handler */
8629      1
8630      1 /*
8631      1 /*+++
8632      1 /*
8633      1 /* FUNCTIONAL DESCRIPTION:
8634      1 /*
8635      1 /*      CTRLC
8636      1 /*
8637      1 /*      AST Routine entered whenever the user strikes CTRL-C.
8638      1 /*      The COLLECTION_END routine is called to begin termination
8639      1 /*      of the Monitor request. Also, the CTRLCZ_HIT bit is set,
8640      1 /*      and the PROMPT bit is set to indicate a MONITOR> prompt
8641      1 /*      is desired.
8642      1 /*
8643      1 /* INPUTS:
8644      1 /*
8645      1 /*      None
8646      1 /*
8647      1 /* OUTPUTS:
8648      1 /*
8649      1 /*      None
8650      1 /*
8651      1 /* ROUTINE VALUE:
8652      1 /*
8653      1 /*      SSS_NORMAL
8654      1 /*
8655      1 /*--
8656      1 /*/
8657      1
8658      1 /*
8659      1 /*      ┌──────────────────────────────────────────────────────────────────────────────────┐
8660      1 /*      │                                                                                      │
8661      1 /*      │                                LOCAL STORAGE                                │
8662      1 /*      │                                                                                      │
8663      1 /*      └──────────────────────────────────────────────────────────────────────────────────┘
8664      1 /*/
8665      1
8666      1 Declare
8667      1     COLLECTION_END ENTRY,
8668      1     CTRLCZ_HIT BIT(1) ALIGNED GLOBALREF,
8669      1     PROMPT BIT(1) ALIGNED GLOBALREF,
8670      1     NORMAL FIXED BINARY(31) GLOBALREF;
8671      1
8672      1 CTRLCZ_HIT = YES;
8673      1 PROMPT = YES;
8674      1 CALL COLLECTION_END();
8675      1 RETURN(NORMAL);
8676      1
8677      1 END CTRLC;
8678
```

```
/* Routine to indicate end of collection */
/* YES => CTRL-C or CTRL-Z has been hit */
/* YES => prompt user for another subcommand */
/* MONITOR normal status value */

/* Indicate CTRL-C has been hit */
/* Indicate user wants MONITOR> prompt */
/* Indicate end of collection */
/* Return to caller */
```



```
8679      CTRLZ: Procedure Returns(fixed binary(31));          /* CTRL-Z handler */
8680      1
8681      1  /*
8682      1  /*++
8683      1  /*
8684      1  /* FUNCTIONAL DESCRIPTION:
8685      1  /*
8686      1  /*      CTRLZ
8687      1  /*
8688      1  /*      AST Routine entered whenever the user strikes CTRL-Z.
8689      1  /*      The COLLECTION_END routine is called to begin termination
8690      1  /*      of the Monitor request. Also, the CTRLZ_HIT bit is set,
8691      1  /*      and the PROMPT bit is set to 0 to indicate a MONITOR> prompt
8692      1  /*      is NOT desired.
8693      1  /*
8694      1  /* INPUTS:
8695      1  /*
8696      1  /*      None
8697      1  /*
8698      1  /* OUTPUTS:
8699      1  /*
8700      1  /*      None
8701      1  /*
8702      1  /* ROUTINE VALUE:
8703      1  /*
8704      1  /*      $$$_NORMAL
8705      1  /*
8706      1  /*--
8707      1  /*/
8708      1
8709      1  /*
8710      1  /*
8711      1  /*
8712      1  /*
8713      1  /*
8714      1  /*
8715      1  /*
8716      1  /*
8717      1  Declare
8718      1  COLLECTION_END ENTRY,
8719      1  COMMAND_FILE FILE GLOBALREF,
8720      1  CTRLZ_HIT BIT(1) ALIGNED GLOBALREF,
8721      1  CTRLZ_HIT BIT(1) ALIGNED GLOBALREF,
8722      1  PROMPT BIT(1) ALIGNED GLOBALREF,
8723      1  EXECUTE BIT(1) ALIGNED GLOBALREF,
8724      1  NORMAL FIXED BINARY(31) GLOBALREF;
8725      1
8726      1  CTRLZ_HIT = YES;
8727      1  CTRLZ_HIT = YES;
8728      1  PROMPT = NO;
8729      1  IF (EXECUTE = YES) THEN DO;
8730      2  CLOSE FILE(COMMAND_FILE);
8731      2  EXECUTE = NO;
8732      2  END;
8733      1  CALL COLLECTION_END();
8734      1  RETURN(NORMAL);

/* Routine to indicate end of collection */
/* File reference for the execute command file */
/* YES => CTRL-Z has been hit */
/* YES => CTRL-C or CTRL-Z has been hit */
/* YES => prompt user for another subcommand */
/* YES => read another execute command file subcommand */
/* MONITOR normal status value */

/* Indicate CTRL-Z has been hit */
/* Indicate CTRL-Z has been hit */
/* Indicate user does NOT want a MONITOR> prompt */
/* If there is an execute command file open, */
/* close the execute command file and */
/* indicate no more execute subcommands to be done. */

/* Indicate end of collection */
/* Return to caller */
```

EXECUTE_REQUEST
V04-000

J 12
16-SEP-1984 02:15:51
5-SEP-1984 15:10:53

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PLI;1 (66)

Page 84

8735 1
8736 1
8737
END CTRLZ;

SHO
V04-


```
8738 CTRLW: Procedure Returns(fixed binary(31));          /* CTRL-W (display screen refresh) handler */
8739
8740 /*
8741 /*++
8742 /*
8743 /* FUNCTIONAL DESCRIPTION:
8744 /*
8745 /*     CTRLW
8746 /*
8747 /*     AST routine, entered whenever the user strikes CTRL-W.
8748 /*     Sets the Refresh Event Flag to indicate a new display
8749 /*     event (including template) is desired.
8750 /*
8751 /* INPUTS:
8752 /*
8753 /*     None
8754 /*
8755 /* OUTPUTS:
8756 /*
8757 /*     None
8758 /*
8759 /* ROUTINE VALUE:
8760 /*
8761 /*     $$$_NORMAL
8762 /*
8763 /*--
8764 /*/
8765
8766 /*
8767 /*
8768 /*
8769 /*
8770 /*
8771 /*
8772 /*/
8773
8774 %INCLUDE          SYS$SETEF;                          /* $SETEF system service */
8780
8781 Declare
8782   REFR_EV_FLAG  FIXED BINARY(31) GLOBALREF VALUE,      /* Refresh event flag */
8783   NORMAL        FIXED BINARY(31) GLOBALREF,            /* MONITOR normal status value */
8784   CALL          FIXED BINARY(31);                      /* Holds function value (return status) of called routines */
8785
8786 CALL = SYS$SETEF(REFR_EV_FLAG);                          /* Cause refresh display event to occur */
8787 RETURN(NORMAL);                                          /* Return to caller */
8788
8789 END CTRLW;
8790
```

```
8791 WRITE_HEADER: Procedure Returns(fixed binary(31));          /* Write recording file header record */
8792 : 1                                                         /* ... and system information record */
8793 : 1
8794 : 1 /*
8795 : 1 /*+++
8796 : 1 /*
8797 : 1 /* FUNCTIONAL DESCRIPTION:
8798 : 1 /*
8799 : 1 /*     WRITE_HEADER
8800 : 1 /*
8801 : 1 /*     Called by the CLASS_COLLECT routine to write the first 2
8802 : 1 /*     records of the recording file (File Header Record and
8803 : 1 /*     System Information Record). Called once per Monitor
8804 : 1 /*     request before any class records are written.
8805 : 1 /*
8806 : 1 /* INPUTS:
8807 : 1 /*
8808 : 1 /*     None
8809 : 1 /*
8810 : 1 /* OUTPUTS:
8811 : 1 /*
8812 : 1 /*     None
8813 : 1 /*
8814 : 1 /* ROUTINE VALUE:
8815 : 1 /*
8816 : 1 /*     $$$_NORMAL, or failing MONITOR status code.
8817 : 1 /*
8818 : 1 /*--
8819 : 1 /*/
8820 : 1
8821 : 1 /*
8822 : 1 /*
8823 : 1 /*
8824 : 1 /*
8825 : 1 /*
8826 : 1 /*
8827 : 1 /*
8828 : 1
8829 : 1 XINCLUDE          MONDEF;          /* Monitor utility structure definitions */
9597 : 1
9598 : 1 Declare
9599 : 1 WRITE_RECORD ENTRY (ANY) RETURNS(FIXED BINARY(31));      /* Routine to write a rec to the recording file */
9600 : 1
9601 : 1 Declare
9602 : 1 CALL          FIXED BINARY(31),          /* Holds function value (return status) of called ro
9603 : 1 STATUS        BIT(1) BASED(ADDR(CALL)),      /* Low-order status bit for called routines */
9604 : 1 NORMAL        FIXED BINARY(31) GLOBALREF,    /* MONITOR normal status value */
9605 : 1 SPTR          POINTER GLOBALREF,            /* Pointer to SYI (System Information Area) */
9606 : 1 MRBPTR        POINTER GLOBALREF,            /* Pointer to MRB (Monitor Request Block) */
9607 : 1 M             POINTER DEFINED(MRBPTR),        /* Synonym for MRBPTR */
9608 : 1 H             POINTER,                      /* Pointer to record file header */
9609 : 1 HEADER TYPE   FIXED BINARY(15) GLOBALREF,    /* Type for MONITOR recording file header */
9610 : 1 ST_LEVEL_CUR  CHAR(8) GLOBALREF,            /* Current MONITOR recording file structure level */
9611 : 1 ST_LEVEL_PB   CHAR(8) GLOBALREF,            /* MONITOR recording file structure level from input
9612 : 1 REVLEVELS     CHAR(128) GLOBALREF,          /* Revision levels used by classes for this request
9613 : 1 REVCLSBITS    BIT(128) GLOBALREF,          /* Bits for classes recorded at rev level 0 */
```


EXECUTE_REQUEST
V04-000

M 12
16-SEP-1984 02:15:53
5-SEP-1984 15:10:53

VAX-11 PL/I X2.1-273
ISK\$VMSMASTER:[MONITOR.SRC]REQUEST.PLI;1 (68)

Page 87

```
9614 1 1 COMM_D BASED(M->MRBSA_COMMENT),
9615 1 2 L FIXED BINARY(15),
9616 1 2 TC CHAR(2),
9617 1 2 A POINTER,
9618 1 COMMENT CHAR(COMM_D.L) BASED(COMM_D.A),
9619 1 1 REC_DESCR,
9620 1 2 L FIXED BINARY(31),
9621 1 2 A POINTER;
9622 1
```

```
/* Descriptor for user's comment string */
/* Length */
/* Type and class */
/* Address */
/* User-specified comment string */
/* Record descriptor */
/* Length */
/* Address */
```

SHOC
V04-

```
9623 1 ALLOCATE FILE_HDR SET (H); /* Allocate file header space */
9624 1
9625 1 H->MNR_HDR$B_TYPE = UNSPEC(HEADER_TYPE); /* Load header type code */
9626 1 H->MNR_HDR$V_FILLER = '0'B; /* Clear all unused flags */
9627 1 H->MNR_HDR$Q_BEGINNING = M->MRB$Q_BEGINNING; /* Load beginning time */
9628 1 H->MNR_HDR$Q_ENDING = '0'B; /* Indicate no ending time yet */
9629 1 H->MNR_HDR$I_INTERVAL = M->MRB$I_INTERVAL; /* Load interval value */
9630 1 H->MNR_HDR$O_REVOCLSBITS = REVOC$BITS; /* Load bits for classes recorded at rev level 0 */
9631 1 H->MNR_HDR$I_RECCT = 0; /* Indicate no records yet */
9632 1 IF M->MRB$V_PLAYBACK /* If a playback request, */
9633 1 THEN H->MNR_HDR$I_LEVEL = ST_LEVEL_PB; /* then load playback recording file structure level */
9634 1 ELSE H->MNR_HDR$I_LEVEL = ST_LEVEL_CUR; /* else load current recording file structure level */
9635 1
9636 1 IF M->MRB$A_COMMENT = NULL() /* If no comment string specified, */
9637 1 THEN DO;
9638 2 H->MNR_HDR$I_COMMENT = ' '; /* Load a string of blanks */
9639 2 H->MNR_HDR$I_COMLEN = 0; /* ... and a length of 0 */
9640 2 END;
9641 1
9642 1 ELSE DO; /* Comment string is specified */
9643 2 H->MNR_HDR$I_COMMENT = COMMENT; /* Load user's comment string */
9644 2 H->MNR_HDR$I_COMLEN = COMM D.L; /* ... and its actual length */
9645 2 IF H->MNR_HDR$I_COMLEN > MNR_HDR$K_MAXCOMLEN /* Minimize actual length with ... */
9646 2 THEN H->MNR_HDR$I_COMLEN = MNR_HDR$K_MAXCOMLEN; /* ... max comment length */
9647 2 END;
9648 1
9649 1 H->MNR_HDR$O_CLASSBITS = M->MRB$O_CLASSBITS; /* Load class bit string */
9650 1 H->MNR_HDR$I_REVLEVELS = REVLEVELS; /* Load revision levels used by this request */
9651 1
9652 1 /*
9653 1 /* Write file header record
9654 1 */
9655 1
9656 1 REC_DESCR.L = MNR_HDR$K_SIZE; /* Load up length */
9657 1 REC_DESCR.A = H; /* ... and address of record for write */
9658 1 CALL = WRITE_RECORD(REC_DESCR); /* Write the file header record */
9659 1 FREE H->FILE_HDR; /* Free file header space */
9660 1 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check WRITE_RECORD call */
9661 1
9662 1 /*
9663 1 /* Write system information record
9664 1 */
9665 1
9666 1 REC_DESCR.L = MNR_SYISK_SIZE; /* Load up length */
9667 1 REC_DESCR.A = SPTR; /* ... and address of record for write */
9668 1 CALL = WRITE_RECORD(REC_DESCR); /* Write the system info record */
9669 1 IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check WRITE_RECORD call */
9670 1
9671 1 RETURN(NORMAL); /* Return to caller */
9672 1 END WRITE_HEADER;
9673 1
```



```
9674 WRITE_RECORD: Procedure (RECORD_DESC)
9675 Returns(fixed binary(31));
9676
9677 /*
9678 /*++
9679 /*
9680 /* FUNCTIONAL DESCRIPTION:
9681 /*
9682 /*     WRITE_RECORD
9683 /*
9684 /*     Called by the WRITE_HEADER and CLASS_COLLECT routines to
9685 /*     write a single record to the recording file. If a flush
9686 /*     has been indicated, it is performed.
9687 /*
9688 /* INPUTS:
9689 /*
9690 /*     Address of a string descriptor describing the record to be written.
9691 /*
9692 /* IMPLICIT INPUTS:
9693 /*
9694 /*     FLUSH_IND -- Flush indicator. If set, perform an RMS flush operation
9695 /*     to "checkpoint" the recording file.
9696 /*
9697 /* OUTPUTS:
9698 /*
9699 /*     None
9700 /*
9701 /* IMPLICIT OUTPUTS:
9702 /*
9703 /*     RECCT incremented by 1.
9704 /*
9705 /* ROUTINE VALUE:
9706 /*
9707 /*     $$$_NORMAL
9708 /*
9709 /*--
9710 /*/
9711
```

```
9712 | 1 /*
9713 | 1 /*
9714 | 1 /*
9715 | 1 /*
9716 | 1 /*
9717 | 1 /*
9718 | 1 /*
9719 | 1 /*
9720 | 1 Declare
9721 | 1     NORMAL          FIXED BINARY(31) GLOBALREF,      /* MONITOR normal status value */
9722 | 1     RECCT          FIXED BINARY(31) GLOBALREF,      /* Count of records written to record file */
9723 | 1     FLUSH_IND      BIT(1) ALIGNED GLOBALREF;        /* Flush indicator; YES => perform FLUSH */
9724 | 1
9725 | 1 Declare
9726 | 1     1 RECORD_DESC,          /* Record descriptor */
9727 | 1     2 RECORD_LEN          FIXED BINARY(31),          /* Record length */
9728 | 1     2 RECORD_PTR          POINTER,                  /* Record pointer */
9729 | 1     RECORD_DATA          CHAR(RECORD_LEN) BASED(RECORD_PTR); /* Record data */
9730 | 1
9731 | 1 Declare
9732 | 1     RECORD_FILE          FILE RECORD;                /* Monitor Record File */
9733 | 1
9734 | 1 WRITE FILE(RECORD_FILE) FROM(RECORD_DATA);          /* Write a record */
9735 | 1 RECCT = RECCT + 1;                                  /* Count it */
9736 | 1
9737 | 1 IF FLUSH_IND                                          /* If flush indicated for this write, */
9738 | 1     THEN DO;
9739 | 2     CALL FLUSH(RECORD_FILE);                        /* Flush record file to checkpoint collected data */
9740 | 2     FLUSH_IND = NO;                                /* Indicate flush not required */
9741 | 2     END;
9742 | 1
9743 | 1 RETURN(NORMAL);                                     /* Return */
9744 | 1 END WRITE_RECORD;
9745 | 1
```

LOCAL STORAGE


```
9746 READ_INPUT: Procedure (SKIP_IND);                                /* Routine to read a record from the /INPUT file */
9747 1
9748 1 /*+++
9749 1 /*
9750 1 /* FUNCTIONAL DESCRIPTION:
9751 1 /*
9752 1 /* READ_INPUT
9753 1 /*
9754 1 /* This routine reads the /INPUT (playback) file until a
9755 1 /* record of the desired type is found, or until end-of-file
9756 1 /* is reached. The following categories of record types exist:
9757 1 /*
9758 1 /*          Types 0 - 127:      Class record
9759 1 /*          Types 128 - 191:    DIGITAL control record
9760 1 /*          Types 192 - 255:    Customer control record
9761 1 /*
9762 1 /* A class record is always desired. A customer control record
9763 1 /* is never desired. A DIGITAL control record can be desired
9764 1 /* or not, depending on the input parameter SKIP_IND.
9765 1 /*
9766 1 /* INPUTS:
9767 1 /*
9768 1 /* SKIP_IND -- a binary longword value indicating whether or not
9769 1 /* to skip past DIGITAL control records. If
9770 1 /* SKIP_IND is 0, DIGITAL control records
9771 1 /* are desired, and will not be skipped.
9772 1 /* Otherwise, they are skipped.
9773 1 /*
9774 1 /* IMPLICIT INPUTS:
9775 1 /*
9776 1 /* MCAPTR -- Pointer to Monitor Communication Area
9777 1 /* INPUT_CPTR -- Pointer to /INPUT file buffer
9778 1 /*
9779 1 /* OUTPUTS:
9780 1 /*
9781 1 /* None
9782 1 /*
9783 1 /* IMPLICIT OUTPUTS:
9784 1 /*
9785 1 /* MCASL_INPUT_LEN is updated to indicate the length of the record
9786 1 /* currently in the input buffer.
9787 1 /*
9788 1 /* MCASV_EOF is set if end-of-file is reached.
9789 1 /*
9790 1 /* ROUTINE VALUE:
9791 1 /*
9792 1 /* None
9793 1 /*
9794 1 /* SIDE EFFECTS:
9795 1 /*
9796 1 /* /INPUT file (INPUT_FILE) is advanced to the desired record.
9797 1 /*
9798 1 /*/
9799 1
```

```
9800 | 1 | /*
9801 | 1 | /*
9802 | 1 | /*
9803 | 1 | /*
9804 | 1 | /*
9805 | 1 | /*
9806 | 1 | /*
9807 | 1 | /*
9808 | 1 | %INCLUDE MONDEF; /* Monitor utility structure definitions */
10576 | 1 |
10577 | 1 | Declare
10578 | 1 | MAX_REC_SIZE FIXED BINARY(31) GLOBALREF VALUE, /* Max record size for PLAYBACK & RECORD files */
10579 | 1 | MCAPTR POINTER GLOBALREF, /* Pointer to MCA (Monitor Communication Area) */
10580 | 1 | MC POINTER DEFINED(MCAPTR), /* Synonym for MCAPTR */
10581 | 1 | INPUT_CPTR POINTER GLOBALREF, /* Ptr to input buffer count word */
10582 | 1 | INPUT_DATA CHAR(MAX_REC_SIZE) VARYING BASED(INPUT_CPTR); /* Playback file input buffer */
10583 | 1 |
10584 | 1 | Declare
10585 | 1 | SKIP_IND FIXED BINARY(31), /* Skip indicator; non-zero => skip DIGITAL control
10586 | 1 | DESIRED_TYPE BIT(1) ALIGNED, /* YES => desired record type found */
10587 | 1 | 1 RECORD_TYPE BASED(MC->MCA$A_INPUT_PTR), /* Record type field of input record */
10588 | 1 | 2 FILLER BIT(6),
10589 | 1 | 2 BIT6 BIT(1),
10590 | 1 | 2 BIT7 BIT(1);
10591 | 1 |
10592 | 1 | Declare
10593 | 1 | INPUT_FILE FILE RECORD INPUT; /* Monitor Input (Playback) File */
10594 | 1 |
10595 | 1 | DESIRED_TYPE = NO; /* Don't have desired type yet */
10596 | 1 | DO WHILE (^ MC->MCA$V_EOF & ^ DESIRED_TYPE); /* Stop reading when hit EOF or desired rec found */
10597 | 2 | READ FILE(INPUT_FILE) INTO(INPUT_DATA); /* Read a record from the input file */
10598 | 2 | IF BIT7 = NO /* If high-order bit of record type off, */
10599 | 2 | THEN DESIRED_TYPE = YES; /* then we found a desired type (class record) */
10600 | 2 | ELSE IF SKIP_IND = 0 & BIT6 = NO /* If caller wants a DIGITAL control rec, */
10601 | 2 | THEN DESIRED_TYPE = YES; /* and it is present, let him have it */
10602 | 2 | END;
10603 | 1 |
10604 | 1 | MC->MCA$L_INPUT_LEN = LENGTH(INPUT_DATA); /* Establish length of input */
10605 | 1 |
10606 | 1 | RETURN; /* Return */
10607 | 1 | END READ_INPUT;
```

COMMAND LINE

PLI/LIS=LIS\$:REQUEST/OBJ=OBJ\$:REQUEST MSRC\$:REQUEST+LIB\$:MONLIB/LIB

F 13

SHO
V04

63

63

63

63

63

0242 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

MONMSG
LIS

REQUEST
LIS

SHODEF
LIS

MONSUB
LIS

PREPOST
LIS

SUMMBUFF
LIS